UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

IA explicable: programación probabilística con PyMC para prevención de blanqueo de capitales

Eduardo Cabria Fuente

Tutor: Alejandro Atienza Ramos

Ponente: Fernando Díez Rubio

MAYO 2021

IA explicable: programación probabilística con PyMC para prevención de blanqueo de capitales

AUTOR: Eduardo Cabria Fuente

TUTOR: Alejandro Atienza Ramos

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Mayo de 2021

Resumen

Este Trabajo Fin de Grado consiste en la implementación de un programa escrito en Python utilizando programación probabilística con el objetivo de conseguir la detección preventiva de blanqueo de capitales en cualquier entidad financiera. El objetivo final es encontrar un modelo PyMC3 único, jerárquico y centrado que permita identificar de manera probabilística las opciones de que una cuenta bancaria sea de carácter *normal* o *criminal*. Además, se pretende determinar qué parámetros o variables de una cuenta pueden dar más información valiosa para saber si una cuenta está cometiendo un delito de lavado de dinero. La idea es que a ese modelo se le introduzca unos datos de entrada formados de muchas transacciones bancarias.

Aplicar este tipo de técnicas, etiquetadas dentro del estimulante mundo del *machine learning*, puede ayudar a aplacar un problema que lleva asolando a gobiernos y entidades financieras durante mucho tiempo. Con ello, podría suponer un importante aumento de los ingresos impositivos de los países que puedan conseguir detener estas prácticas ilícitas.

Como marco de trabajo para la programación probabilística se ha usado el framework PyMC3. PyMC3 ofrece muchas ventajas, entre las que destaca su ayuda a resolver problemas básicos de inferencia y predicción estadística bayesiana mediante la descripción de modelos probabilísticos con una sintaxis potente, legible e intuitiva.

Por un lado el avance en los sistemas informáticos de los últimos años y por otro, el desarrollo de métodos matemáticos como la cadena de Markov, que se pueden adaptar para calcular la distribución *a posteriori* (referente al Teorema de Bayes) de cualquier modelo, además del avance en las técnicas de muestreo de los métodos de Monte Carlo; han transformado radicalmente el mundo del análisis de datos bayesiano.

Palabras clave

PyMC3, Programación Probabilística, inferencia bayesiana, modelo, distribución a priori, likelihood, distribución a posteriori, blanqueo de capitales, transacciones, cuentas normales, cuentas criminales, base de datos.

Abstract

This Bachelor Thesis consists of the implementation of a program written in Python using probabilistic programming with the aim of achieving the preventive detection of money laundering in any financial institution. The project goal is to find a unique, hierarchical and centered PyMC3 model that allows us to probabilistically identify the options of a bank account being of a *normal* or *criminal* nature. In addition, it is also intended to determine which parameters or variables of an account can give us more valuable information to know if an account is committing a money laundering crime. The idea is that this model is fed with input data composed of many bank transactions.

Applying this type of techniques, labeled inside the stimulating world of *machine learning*, can help to placate a problem that has been dealing with governments and financial institutions for a long time. In doing so, it could lead to a significant increase in tax revenues for countries that can succeed in stopping these illicit practices.

The PyMC3 framework has been used as a framework for probabilistic programming. PyMC3 offers many advantages, including helping to solve basic Bayesian statistical and prediction problems by describing probabilistic models with a powerful, simple and intuitive syntax.

On the one hand, the progress in computer systems in recent years and on the other hand, the development of mathematical methods such as Markov chain, which can be adapted to calculate the *posterior* distribution (referring to Bayes' Theorem) of any model, in addition to the progress in sampling techniques of Monte Carlo methods, have radically transformed the world of Bayesian data analysis.

Keywords

PyMC3, Probabilistic Programming, Bayesian inference, model, prior distribution, likelihood, posterior distribution, money laundering, transactions, normal accounts, criminal accounts, database.

Agradecimientos

A todos mis seres queridos que me han acompañado, apoyado y animado en este camino.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1. Motivación

El contexto general del proyecto en el que se enmarca este trabajo de fin de grado es la lucha contra el blanqueo de capitales. Este es un asunto que preocupa a la Unión Europea (UE) desde hace tiempo, dado que no consigue controlar el problema que tiene con el lavado de dinero dentro de sus fronteras. A pesar de que los máximos responsables de estas parcelas en el continente hayan hecho numerosas revisiones en la política y detección de este tipo de fraudes, todavía hay un continuo flujo de dinero de origen dudoso. De hecho, en estos momentos se está revisando la directiva europea actual para actualizarla en la sexta enmienda contra el blanqueo de capitales que se redacta en Europa, de la que se pretende que sea un enorme y decisivo paso en determinados ámbitos de la ley para frenar esta sangría de dinero ilegal [1]. La solución que se propone desde la UE y que están estudiando los diferentes ministros de la eurozona implica crear, como medidas más importantes, una regulación común y una autoridad central anti-blanqueo, entre otras medidas [2]. Se quiere cambiar la vigente regulación a causa de que, en primer lugar, deja mucho espacio para que cada Gobierno la adapte de acuerdo con sus intereses y necesidades. La consecuencia de este hecho es la dificultad de controlar las transacciones de lavado de dinero, porque dentro de la UE existe mucha diversidad reglamentaria. Una vez el dinero entra en el mercado único, se puede mover con mucha facilidad, y este es el verdadero problema. Está demostrado que países pequeños como Malta, Chipre o Bulgaria son realmente un agujero para estas operativas [3]. Además, muchas de las grandes fortunas tienen detrás equipos de asesoramiento en materia fiscal que les permiten hacer cantidad de operativas para desviar fondos o evadir impuestos sin que las autoridades recaudatorias puedan hacer nada debido al complejo entramado que forman con los instrumentos financieros que poseen (y les permiten ciertos estados). Esto último se suele llamar *ingeniería fiscal*.

Por todas estas razones, se pretende crear un modelo probabilístico usando tecnologías de *machine learning*, el análisis de datos y herramientas de estadística y probabilidad, todas ellas modernas y pioneras las cuales se tiene actualmente a nuestra disposición. La intención también es que se sustente con casos reales como el que se propone en este trabajo final de grado, lo cual sería una posible solución para este rompecabezas que lleva asolando Europa y el mundo en general desde que los países tienen sus monedas reguladas. El objetivo principal no sería más que acceder a recuperar esas grandes sumas económicas para las arcas públicas de los países miembros de la UE, que permitiría luchar contra la crisis de déficit y de deuda a la que se enfrenta nuestro continente por la crisis de la COVID-19 de una manera más holgada.

1.2. Objetivos

El objetivo principal de este trabajo es desarrollar un *notebook* en *jupyter*, en el cual se describa un modelo único, jerárquico y centrado utilizando programación probabilística (en este caso, utilizando la herramienta PyMC3) el cual sea capaz de detectar indicios de un posible blanqueo de capitales mediante la clasificación de cuentas bancarias en *normales* y *criminales* evaluando la probabilidad condicionada.

El instrumento por el que gira este proyecto, PyMC3, se define como un nuevo framework de código abierto desarrollado en Python con una sintaxis intuitiva, simple, legible y, sobre todo, potente que se acerca a la sintaxis natural que usan los estadísticos para describir los diferentes modelos. Se engloba dentro de la llamada programación probabilística (PP) que permite la especificación en código flexible de los modelos estadísticos bayesianos. PyMC3 es el 'alfabeto' con el que se corren los modelos de este proyecto para realizar diferentes simulaciones.

En definitiva, se pretende que dicho *notebook* sea claro e intuitivo, con explicaciones detalladas de cada paso. Hay que ajustar y modelar el programa con datos de entrada que se generan de manera aleatoria sobre transacciones de cuentas bancarias, consiguiendo unos datos de salida que puedan indicar si esas cuentas pueden tener evidencias de ser ilegales. Todo esto juntado con un caso práctico que se haya dado en el mundo real con cuentas de comportamientos *normales* e inferir la probabilidad de detectar un caso fraudulento. Habrá que tener muy en cuenta en cómo se modifican esas variables y parámetros de las cuentas bancarias que se pasan como datos de entrada después de muestrear el modelo con PyMC3, con el objetivo de sacar conclusiones posteriores.

Para conseguir este objetivo, se estudiará a fondo la herramienta PyMC3, tanto su fundamento teórico como sus distintas posibles funcionalidades, configurando el entorno necesario con las librerías necesarias para su explotación. Aparte, se pretende usar *ArviZ* como librería de visualización de los datos de salida, los cuales se producirán después del paso de la inferencia bayesiana con PyMC3, con el fin de que se puedan sacar esas conclusiones de manera más intuitiva.

Adicionalmente, el segundo gran objetivo que tiene este proyecto es fundamentalmente didáctico. Se trata de desarrollar y explicar los conceptos de la base teórica de PyMC3 y del análisis de datos, que cada vez tiene más relevancia tanto para grandes y medianas empresas como para gobiernos nacionales, con el ánimo de entender mejor el mundo que nos rodea con herramientas estadísticas/matemáticas, a través de la programación probabilística y el paradigma bayesiano. Se trata de dar un enfoque no solo de determinar y llegar a una solución del problema original planteado, sino también de realizar una evaluación de la librería y las técnicas probabilísticas, con lo que ello engloba, a la hora de la resolución del problema.

2 Estado del arte

2.1. Introducción a los conceptos principales

2.1.1. Programación Probabilística

La programación probabilística (PP) está diseñada para describir modelos y distribuciones de probabilidad de forma que se puedan realizar inferencias probabilísticas mediante la programación, o a su vez, construyendo programas para desarrollar dichos modelos. La aplicación de una distribución, una consulta o un algoritmo de inferencia puede ser una tarea que lleve mucho tiempo, incluso para alguien que sea experto en estas áreas. La automatización de este proceso a través de un lenguaje de PP permite una exploración más amplia del espacio de diseño sin el esfuerzo adicional que se requiere para utilizar un enfoque tradicional, haciendo una clara separación entre la creación del modelo y el proceso de inferencia.

La PP en Python confiere una serie de ventajas respecto a otros lenguajes que usan los estadísticos, entre las que se incluyen la compatibilidad con múltiples plataformas, una sintaxis expresiva pero limpia y legible, una fácil integración con otras bibliotecas científicas y la posibilidad de ampliación a través de C, C++, Fortran o Cython [4]. Estas características hacen que sea relativamente sencillo escribir y utilizar distribuciones estadísticas personalizadas, muestreadores y funciones de transformación, como requiere el análisis bayesiano.

B. Cronin [5] describe, de un modo muy motivador el concepto de la programación probabilística: *"Otra forma de pensar en esto: a diferencia de un programa tradicional, que sólo se ejecuta en la dirección hacia adelante, un programa probabilístico se ejecuta tanto en la dirección hacia adelante como hacia atrás. Se ejecuta hacia adelante para calcular las consecuencias de las suposiciones que contiene sobre el mundo (es decir, el espacio del modelo que representa), pero también se ejecuta hacia atrás a partir de los datos para restringir las posibles explicaciones. En la práctica, muchos sistemas de programación probabilística intercalan hábilmente estas operaciones hacia delante y hacia atrás para encontrar las mejores explicaciones."*.

Unos ejemplos de los más importantes lenguajes de programación probabilística, aunque existen decenas, pueden ser: *Pyro*, *Hakaru*, *Stan* y el ya mencionado, PyMC3.

2.1.2. PyMC3

PyMC nació en 2003 desarrollado por una comunidad de científicos vinculados a la Universidad de Georgia en EE. UU., como un esfuerzo por generalizar el proceso de construcción del algoritmo hamiltoniano de Monte Carlo (HMC), y con el objetivo de hacer más accesible el método de Monte Carlo basado en cadenas de Markov (MCMC) a los científicos aplicados. En 2006, tres científicos analistas estadounidenses, David Huard, Anand Patel y Chris Fonnesbeck, se unieron para dar un impulso importante a la primera versión del framework creando PyMC2.0. Esta versión se caracterizaba por conseguir más flexibilidad, mejor rendimiento y una mejor experiencia para el usuario final. PyMC 2.3 se publicó el 31 de octubre de 2013 incluyendo la compatibilidad con Python 3, la mejora de los gráficos de resumen, además de otras mejoras y correcciones de errores y optimizaciones importantes. Fue cuando despegó y se popularizó en la comunidad informática y estadística

internacional. La versión actual del framework, que es la tercera, oficialmente lanzada en enero de 2017, amplió el abanico de personas en el equipo de desarrollo trabajando en él. Además, se llevó a cabo la adopción de *Theano* como desarrollo back-end para poder implementar muestreadores MCMC basados en el gradiente [6]. Actualmente la última versión del framework es la 3.11.2.

Entre las características principales que se pueden destacar de PyMC3, hay que mencionar que cuenta con algoritmos MCMC de última generación, como el muestreador sin vuelta (NUTS), una variante autoajustable de Hamiltonian Monte Carlo (HMC). PyMC3 pertenece a un reducido grupo de paquetes de PP que ofrecen HMC junto con *Stan*, también en Python, y el paquete *LaplacesDemon* para R.

El conjunto de características que tiene PyMC3 se puede detallar en:

- Se ajusta a los modelos estadísticos bayesianos haciendo uso de cadenas de Markov Monte Carlo, inferencia variacional y otros algoritmos relacionados.
- Se incluye un gran conjunto de distribuciones estadísticas bien documentadas. PyMC3 también tiene la capacidad de crear resúmenes que incluyen tablas y gráficos, y las *traces* que se generan después de la inferencia pueden ser guardados en el disco como archivo de texto simple, SQLite o cuadros de datos de pandas.
- Es un framework extensible, esto es, incorpora fácilmente métodos de pasos personalizados y distribuciones de probabilidad inusuales o poco frecuentes, y los bucles MCMC generados pueden ser incorporados en programas más grandes, siendo posible analizar los resultados con toda la potencia de Python.

2.2. Fundamentos teóricos que sustentan a PyMC3

2.2.1. El universo bayesiano

Para los estadísticos pioneros del siglo XVIII como Bernoulli, Bayes y Laplace, el concepto de probabilidad representaba un grado de creencia o verosimilitud asignado a un resultado, es decir, cuánto pensaban que algo era cierto, según las pruebas/evidencias de que se disponía. Sin embargo, para los estudiosos del siglo XIX, esta idea parecía demasiado vaga y subjetiva para ser la base de una teoría matemática rigurosa. Así que redefinieron la probabilidad como la frecuencia relativa a largo plazo con la que ocurría un evento, dadas (infinitamente) muchas pruebas repetidas (experimentales). Dado que las frecuencias pueden medirse, la probabilidad se consideró como un instrumento objetivo para tratar los fenómenos aleatorios.

Estos tipos de fenómenos se pueden explicar con un sencillo modelo aplicado al tema principal del proyecto:

- Sabiendo que un banco tiene cierto porcentaje de cuentas fraudulentas, por ejemplo el 1%, es posible calcular la probabilidad de que una cuenta sea en efecto *criminal* de entre 100 cuenta, 99 cuentas, 98 cuentas… y así sucesivamente, al extraer 100 cuentas al azar de nuestra población.
- Pero, desde otra perspectiva, si se tiene en cuenta los datos de cuántas cuentas son ilegales, ¿cómo se puede calcular ese porcentaje? ¿Y cómo se puede estar seguro de

tal porcentaje? ¿O qué probabilidad hay de que se produzca un determinado porcentaje de cuentas fraudulentas?

La ciencia continuamente suele plantearse preguntas del segundo punto, y el pensamiento bayesiano ofrece un marco maravilloso para responder ambos puntos. Esencialmente, el Teorema de Bayes ofrece una forma de pasar de la probabilidad de los datos dado el modelo $P(datos|modelo)$ a la probabilidad del modelo dados los datos $P(modelo|datos)$.

Thomas Bayes fue un ministro presbiteriano y matemático inglés que dio nombre a una de las páginas más importantes de la teoría de la probabilidad en la historia. Publicado póstumamente en 1793, esta proposición plantea unos fundamentos filosóficos potentes junto a unas matemáticas simples. Dado que, en su forma más básica, no es más que una expresión algebraica con tres variables conocidas y una incógnita; y además trabaja con probabilidades condicionales. Este teorema indica la probabilidad de que una hipótesis θ sea verdadera si algún evento *y* ha sucedido, lo que hace que sea muy útil en los casos los cuales se sabe la probabilidad de los efectos dadas las causas, y se quiere saber la probabilidad de las causas dados los efectos [7]. Así el Teorema de Bayes permite ir de uno a otro con suma facilidad.

Además, se pretende explicar de manera detallada el Teorema de Bayes porque la estadística y la mecánica del análisis bayesiano para entender los modelos y las visualizaciones es la base principal y fundamental de PyMC3. Su ecuación se puede encontrar en la *Figura 2-1*.

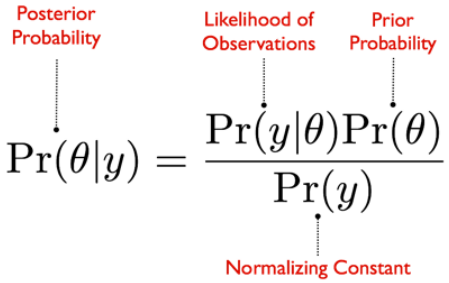


Figura 2-1: Ecuación del Teorema de Bayes. Elaboración propia.

Esta formulación expresa cómo es nuestra creencia sobre el valor de θ, expresada por:

- La distribución o probabilidad *a priori* $P(\theta)$, que es la forma de introducir el conocimiento previo sobre los valores que puede tomar la hipótesis.
- La observación de los datos o también llamada '*likelihood*' $P(y|\theta)$. Esta expresión $P(y|\theta)$ indica la probabilidad de ver los datos *y* dado un θ concreto. A medida que la cantidad de datos aumenta, la *likelihood* tiene cada vez más peso en los resultados.
- La distribución *a posteriori* $P(\theta|y)$, es la consecuencia que se obtiene después de realizar los experimentos bayesianos un cierto número de veces. Se suele definir como la versión actualizada del *a priori* luego de que se han observado, recopilado y agregado los datos adicionales al modelo.

El denominador inocuo $P(y)$, que es la evidencia del modelo o probabilidad marginal. Este valor es prácticamente inofensivo en los modelos, de hecho, en la mayoría de problemas computacionales suele omitirse y se toma como una simple constante de normalización.

Además, no se puede calcular directamente y es en realidad la expresión del numerador integrada sobre todos los θ, como puede verse en la *Figura 2-2*.

$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{\int Pr(y|\theta)Pr(\theta)d\theta}$$

Figura 2-2: Ecuación del Teorema de Bayes, con la parte del denominador integrada. Elaboración propia.

El cálculo de la distribución *a posteriori* $P(\theta|y)$ se denomina el problema de la inferencia, que es uno de los objetivos del análisis bayesiano. Ese cálculo es la tarea básica de cualquier sistema de inferencia probabilística. Toda forma de inferencia que se realice en el mundo que nos rodea, debe enfrentarse con la incertidumbre. Existen tres tipos de incertidumbre con la que se lidia: la ignorancia, debido a los límites de nuestro conocimiento; la aleatoriedad, dada la influencia del azar en todo lo que nos rodea en el día a día; y la vaguedad, debido a la subjetividad que reina en cada uno de nuestros pensamientos [7].

La inferencia bayesiana, en particular, es la filosofía que afirma que para entender la opinión humana (limitada por la ignorancia y la incertidumbre) se debe utilizar el cálculo de la probabilidad como la herramienta más importante para fortalecer nuestras creencias. La inferencia bayesiana combina la experiencia previa (en forma de probabilidad *a priori*) con los datos observados (en forma de *likelihood*) para interpretarlos y encontrar una distribución *a posteriori*. Asimismo, la inferencia bayesiana no garantiza que se pueda dar la respuesta correcta, más bien proporciona una probabilidad de que cada una de un número de respuestas alternativas, sea verdadera. Y luego, se utiliza esta información para encontrar la respuesta que más probablemente sea la correcta.

2.2.2. Métodos de cadenas de Markov Monte Carlo (MCMC)

Para aplicar esta inferencia bayesiana, se muestrea la distribución *a posteriori*. Como no se tiene o no se conoce un método general para muestrear distribuciones de altas dimensiones (ya que la inferencia bayesiana en muchas ocasiones necesita integrar funciones de distribuciones de dimensiones muy elevadas, incluso con miles de parámetros) se usa MCMC. Las siglas MCMC definen la integración de las cadenas de Markov y el método de Monte Carlo.

Las cadenas de Markov fueron introducidas como concepto por el matemático ruso Andréi Markov en 1907. Sirve para calcular la probabilidad de que se dé un evento, única y exclusivamente teniendo en cuenta los datos del evento o estado inmediatamente anterior. Las cadenas de Markov no dejan de ser un tipo de proceso estocástico a tiempo discreto el cual es muy útil para realizar predicciones sobre meteorología, simulaciones al azar, música… etc. De una manera más formal, una cadena de Markov $X_1, \dots X_t$ es una secuencia de variables aleatorias, que no son independientes ni idénticamente distribuidas (i.i.d.) y que cumplen la siguiente propiedad:

$$P[X_{t+1}|X_t, X_{t-1}, X_{t-2}, \dots] = P[X_{t+1}|X_t]$$

Bajo condiciones laxas de regularidad, una cadena de Markov, según vaya avanzando irá convergiendo hacia lo que se denomina distribución estacionaria, sin depender de cuál sea su estado inicial ni del número de estados que la compongan [8]. Normalmente, los resultados generados por las cadenas de Markov no son i.i.d., por lo que se utiliza el teorema central del límite para conseguir valores parecidos a los garantizados a fin de realizar las estimaciones. Además, para que una cadena de Markov converja a una distribución estacionaria, debe cumplir varias condiciones y propiedades. Hay varias características en una cadena de Markov:

- Una cadena de Markov es homogénea si la probabilidad de ir del estado *i* al estado *j* en un paso no depende del tiempo en el que se encuentra la cadena.
- Las transiciones o pasos entre estados en una cadena de Markov se definen como la probabilidad de ir del estado *i* en el tiempo *n* al estado *j* en el tiempo *n+1*.
- Teniendo las probabilidades de transición de un paso (p_{ij}), si se varía los índices sobre el espacio de estados total de la cadena, se obtiene la matriz de probabilidades de transición, cuyas propiedades son:
    1. Debe ser una matriz estocástica cuadrada.

       $A \in \mathbb{R}^{n \times n}$
    2. Ninguno de los números dentro de la matriz debe ser mayor a 1 (entre 0 y 1). $a_{ij} \geq 0$
    3. La suma de las probabilidades debe ser siempre igual a 1.

       $$\sum_{j} a_{ij} = 1$$

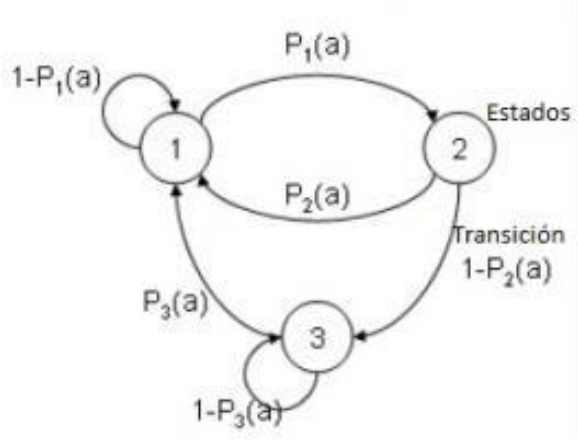


Figura 2-3: Matriz de transición donde se observa el comportamiento estacionario de la cadena de Markov a través de los diferentes estados y transiciones. Extraído de: [9].

Con respecto a la otra pata de MCMC, el método de Monte Carlo fue inventado por Stanislaw Ulam en 1940 mientras trabajaba en el Proyecto Manhattan en Los Álamos, EE. UU. Lo llamaron así porque la base para que funcione bien es un buen generador de números aleatorios, de hecho, generarlos de manera que sea puramente aleatoria es uno de los problemas de computación más complicados que existen. El funcionamiento es el siguiente; con esos números aleatorios se generan múltiples casos como pueden ser, por ejemplo,

infinitas partidas de un juego de azar. Con esos casos es posible hacer muchas pruebas, y con las pruebas, hacer cálculos. El poder del método Monte Carlo reside en que repitiendo algo muchas veces se puede obtener aproximaciones sin hacerlo analíticamente con papel y bolígrafo. Por poner un ejemplo, en el CERN, con ayuda de Monte Carlos crearon un acelerador de partículas ficticio que generando números aleatorios es capaz de recrear las colisiones. Esas colisiones 'falsas' se comparan con las colisiones reales para ver si los modelos son ciertos y así sacar conclusiones con los datos [10].

Entonces, la unión de estos dos términos en los métodos de Markov Monte Carlo es útil porque permite construir cadenas de Markov cuya distribución estacionaria sea la distribución posterior del modelo, la cual se muestrea. Aparte, usando la integración Monte Carlo se tendrá facilidad para tomar estimaciones o medias muestrales de funciones de la cadena para obtener dicha información. En definitiva, en los métodos MCMC el objetivo principal es muestrear de una distribución estacionaria.

Lo primero y más importante de MCMC, aparte de entender su concepto y lo que engloban las siglas, es aprender a cómo construir cadenas con la propiedad de Monte Carlo. Existen una larga familia de algoritmos que aplican MCMC. Todos ellos tienen el mismo objetivo, calcular de la forma más eficiente la distribución *a posteriori*. La mayoría de estos algoritmos siguen unos pasos expresados, de una manera de alto nivel [4], a continuación:

I. Se empieza en la posición actual.

II. Se propone moverse a una nueva posición (investigar los puntos de alrededor).

III. Aceptar o rechazar la nueva posición basándose en la cohesión de la posición a los datos y las distribuciones *a priori* (preguntar si esos puntos están cerca o no de nuestro objetivo).

IV. Si acepta: Desplazar a la nueva posición. Volver al paso 1.

Si no: No trasladarnos a la nueva posición. Volver al paso 1.

V. Después de un gran número de iteraciones, devolver todas las posiciones aceptadas.

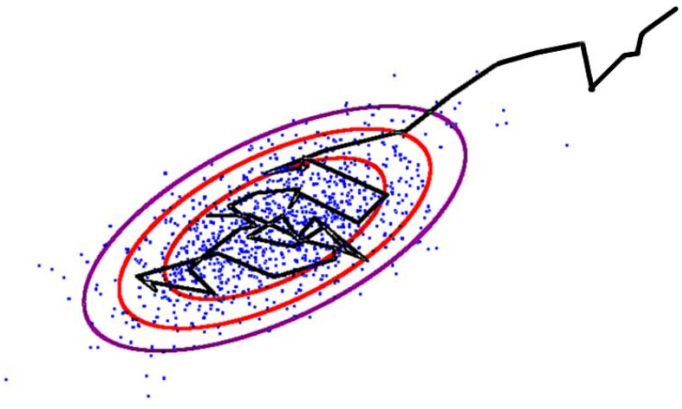


Figura 2-4: Ejemplo gráfico de las iteraciones que puede realizar un algoritmo MCMC. Extraído de: [11].

De esta manera, el algoritmo se mueve en la dirección hacia las regiones donde existen las distribuciones *a posteriori*, y las muestras realizadas se recogen en este proceso una vez que

se consigue llegar a la posterior. Si la posición actual del algoritmo MCMC se encuentra en zonas de probabilidad extremadamente bajas, lo que suele ocurrir cuando el algoritmo comienza, el algoritmo se moverá en posiciones que no sean de la posterior pero que son mejores que todo lo que hay cerca. Por lo tanto, los primeros movimientos no reflejan las posterioridades. Estas posterioridades sí que se reflejan en el pseudocódigo que se ha detallado arriba, ya que solo importa la posición actual (las nuevas posiciones se investigan sólo cerca de la posición actual). A continuación, se detalla concisamente algunos de los algoritmos más populares de MCMC:

- Algoritmo Metropolis-Hastings

  Es el más simple y flexible de los algoritmos MCMC. El muestreo por rechazo es el precursor de este algoritmo, el cual proporciona de una manera muy eficiente la simulación de valores a partir de una variable aleatoria $f(x)$. En este caso, los valores aleatorios se toman de distribuciones escogidas de tal manera que se comporten asintóticamente como valores de la distribución objetivo.

- Algoritmo Hamiltoniano de Monte Carlo (HMC)

  El HMC surgió de la ineficiencia del Algoritmo de Metropolis-Hastings para muchos modelos. En este contexto, y cuando el muestreo se realiza a partir de variables continuas, el HMC puede resultar muy potente. Evita el comportamiento de paso aleatorio entre los estados, al simular un sistema físico gobernado por la dinámica hamiltoniana, evitando distribuciones excesivamente complejas en el proceso. Por tanto, las muestras en el algoritmo HMC se obtienen mediante la simulación de un sistema físico, en el que las partículas se mueven por un marco de altas dimensiones, sujetas a energías potenciales y cinéticas. Utiliza también la eficiencia demostrada del muestreo basado en el gradiente, usado en disciplinas de *redes neuronales* y *machine learning*.

- No U-turn Sampler (NUTS)

  La mayor desventaja del algoritmo HMC es la cantidad de ajustes que se requieren para que el muestreo sea efectivo. Entre los parámetros que deben ser especificados por el usuario está la escala de la distribución del momento, el tamaño del paso de salto para el algoritmo y el número de pasos a dar. Cuando estos parámetros están mal elegidos, el algoritmo HMC sufre de ineficiencia. Un algoritmo MCMC eficiente busca la optimización manteniendo un equilibrio, y aunque el HMC puede ajustarse sobre la marcha, requiere costosas ejecuciones continuadas para hacerlo. Por ello, se creó el algoritmo de muestreo 'sin giro' (NUTS) que ajusta automáticamente los parámetros de tamaño y número de pasos, sin intervención del usuario. Como resultado, NUTS es el algoritmo de muestreo por defecto para variables continuas en PyMC3.

2.2.3. El análisis exploratorio de los datos y la estimación de los parámetros

La estadística predictiva o el análisis exploratorio de los datos (EDA, como se conoce comúnmente por las siglas en inglés) es un enfoque para la realización del análisis de datos que emplea una variedad de técnicas (principalmente gráficas) con el objetivo de [12]:

- Maximizar el conocimiento de un conjunto de datos.
- Descubrir la estructura subyacente.
- Extraer variables importantes.
- Detectar valores 'outlayers'.
- Determinar la configuración óptima de los factores o la estimación precisa de los parámetros.

La estadística predictiva y la inferencia bayesiana pueden ser especialmente eficaces en combinación. Es posible hacer encajar el análisis exploratorio de los datos con el paradigma de la modelización de la probabilidad, ya que el software bayesiano se puede generalizar para dibujar datos y parámetros replicados a partir de las simulaciones predictivas de sus distribuciones posteriores, y éstas a su vez utilizarse para calibrar las visualizaciones y gráficos de la EDA [13].

Tanto el proceso de análisis exploratorio de los datos como, sobre todo, la estimación de los parámetros a la hora de construir nuestro modelo, son elementos muy importantes en PyMC3. Dependiendo del tipo de caso a estudiar, y sobre todo, del tipo de naturaleza y comportamiento de los datos, se tendrá que elegir unos tipos de parámetros u otros para conseguir que la inferencia bayesiana se realice correctamente y conseguir unos datos posteriores acordes al problema enfrentado.

Por lo tanto, para la óptima construcción de los modelos bayesianos, el primer paso es especificar completamente el modelo en términos de distribuciones de probabilidad. Esto consiste en detallar una función de *likelihood* para las observaciones y unas distribuciones *a priori* para todas las variables desconocidas. Para definir estas *a priori* en términos de distribuciones hay que tener en cuenta una serie de consideraciones: si son valores discretos o continuos, cualquier información conocida sobre esa variable o el soporte de ella, es decir, el conjunto de puntos donde la función no es cero. Se utilizarán por lo tanto la gran cantidad de distribuciones probabilísticas que existen en el mundo de la estadística, para asignarlas a las variables y utilizarlas intuitivamente para reflejar que idea se tiene sobre las *a priori*, *likelihood* y *a posteriori*.

Por poner un caso de ejemplo, en el que se intenta modelar una ratio de porcentaje de gente que hace clic en una página web concreta (*A/B test*) se tratará de identificar la forma de la distribución de muestreo de los datos y cómo se describe nuestra incertidumbre sobre los parámetros envueltos. Sin tener ningún tipo de información previa del comportamiento de este caso, nuestra distribución *likelihood* de muestreo será Binomial porque el comportamiento de una persona es hacer clic o no en nuestra web. Así, se obtiene un resultado entre dos posibles al realizar un número determinado de pruebas, y la *a priori* (que es única porque la *likelihood* solo requiere un parámetro) la cual engloba toda nuestra incertidumbre, solo puede ir acotada entre cero y uno (la probabilidad en una distribución binomial no puede ser más grande que uno y no puede ser negativa). Por esto y porque se ha

dicho que no se tiene la información previa sobre este parámetro (igual credibilidad de cero a uno) se utilizaría una distribución uniforme para definirla. Si hubiese algún tipo de observaciones pasadas, esta estimación de los parámetros podría cambiar, con lo que nuestra salida (la distribución *a posteriori*) también se modificaría.

2.2.4. Theano

En el ámbito del *Machine Learning* (ML), *Theano* es una librería de cálculo numérico escrita en Python que permite definir funciones que implican operaciones con *arrays,* funcionando muy bien con problemas que implican álgebra lineal. *Theano* ayuda, por ejemplo, en las fases cuando al implementar unos datos dentro de una red neuronal, o un algoritmo de ML, hay que entrenarlos para conseguir sacar predicciones. Esta es una de las partes más complicadas a nivel teórico del ML. Aplicaciones como *Theano* permite que estos grandes cálculos numéricos que se engloban es estas fases sean más sencillas de modo computacional. *Theano*, en definitiva, es una biblioteca de Python que permite definir, optimizar y evaluar expresiones matemáticas que involucran matrices multidimensionales de manera eficiente [14].

Figura 2-4: Logos de la integración de PyMC3 con Theano. Extraído de: *[15].*

Cuando definimos un modelo en PyMC3, implícitamente construimos una función escrita con *Theano*, en la que se calculan desde el espacio de nuestros parámetros del modelo hasta la densidad de probabilidad de la distribución *a posteriori*. Además, PyMC3 aprovecha *Theano* para codificar de forma transparente los modelos al lenguaje C y compilarlos a código máquina, lo que aumenta significativamente el rendimiento. Al utilizar una GPU, *Theano* realiza cálculos intensivos de datos mucho más rápido que en una CPU [16]. *Theano* utiliza estructuras de datos vectoriales denominados tensores, que están estrechamente integradas con la popular estructura de datos *ndarray* de *NumPy*, y que permiten igualmente la difusión y la indexación avanzada, al igual que las matrices de *NumPy*. Esto hace que PyMC3 use a *Theano* como su back-end computacional. En 2017, los desarrolladores fundadores de *Theano* dejaron de dar mantenimiento a la librería, por lo que los desarrolladores de PyMC se hicieron cargo de ello para extender la vida de PyMC3.

2.3. Aplicaciones en las que se ha empleado PyMC3

Se pueden encontrar ejemplos de uso en el sector financiero (que es el que se ocupa en este proyecto), en marketing digital, en retail, en el ámbito deportivo, en el sector energético, en aseguradoras, industria, salud, ciencia, personas/RRSS, etc. Como puede verse, PyMC3, o en general, la PP se pueden encontrar en alguna aplicación en casi cualquier sector.

El primer caso que se va a presentar es la predicción de los posibles resultados en el Campeonato de Seis Naciones de Rugby [17]. Se usan los resultados de las ediciones de 2014, 2015, 2016, y 2017 a fin de construir un modelo para el próximo torneo de 2018. Se quiere inferir el parámetro oculto que significa la 'fuerza' que tiene cada equipo, para ello se utiliza los parámetros que si existen, las líneas de puntuación de cada equipo y sus resultados en los partidos observados. Además, se conoce que las líneas de puntuación son una medida ruidosa para concluir la fortaleza de cada equipo, por lo que lo ideal es crear un modelo que facilite la cuantificación de nuestra incertidumbre sobre los parámetros que no se aprecian de manera evidente. En definitiva, se pretende cuantificar esa incertidumbre, utilizarla para generar un modelo y que la respuesta sea en forma de distribuciones, no estimaciones puntuales. Después del análisis exploratorio y visuales de los datos y de construir el modelo, el cual es jerárquico, en PyMC3 especificando tanto los parámetros globales y los específicos como la función de *likelihood*, se determina que tanto Inglaterra, Gales e Irlanda tienen las mayores posibilidades de hacerse con el título, detrás de Escocia y Francia que están un peldaño por detrás, e Italia que es última con diferencia. Además, se sacan diferentes métricas para analizar la capacidad atacante y defensiva de cada equipo. En la *Figura 2-5* se muestra visualmente la afirmación anterior, en la que se indica el intervalo más alto de densidad de la posterior (HPD), que se refiere al intervalo más corto que contiene el 95% de la probabilidad de la distribución *a posteriori*.

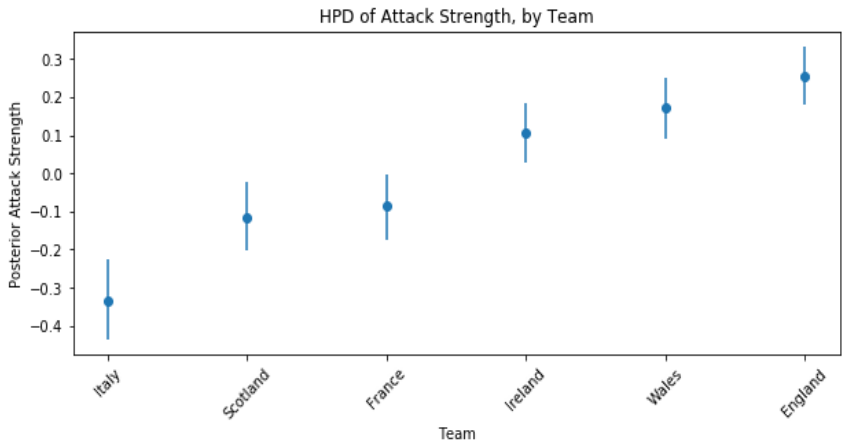


Figura 2-5: Análisis de los pesos de la distribución posterior para la fuerza de ataque de los equipos analizados. Extraído de: [17].

El segundo caso que se presenta consiste en el uso de la inferencia bayesiana para determinar una prueba de emisiones inusuales en los automóviles Volkswagen, noticia que tuvo repercusión a nivel mundial hace cinco años [18]. En este ejemplo de uso de PyMC3, cuyo

código está disponible en GitHub[1], el autor primeramente carga los datos 'en crudo' de emisiones de coches de muchas marcas tomadas de una fuente vía web, y realiza una limpieza y unas exclusiones del dataset principal, además de crear nuevas características en éste que ayuden en el proceso creación del modelo jerárquico. Después, se realiza un análisis exploratorio de los datos para sacar conclusiones de ellos (elegir el parámetro dependiente y las características independientes) y afrontar de una manera más correcta la creación del modelo jerárquico que va a permitir predecir el comportamiento de las distintas emisiones en los coches. En resumen, parece que Volkswagen tiene un parámetro inusualmente alto y ajustado que explica sus emisiones de las partículas de NOx.

En el tercer caso, se presenta el desarrollo de modelos en PyMC3 en una de las cuestiones que más ha afectado a la población mundial este último año, la pandemia de la COVID-19. En este artículo [19] se pretende inferir la relación de la propagación del virus por la población y las medidas propuestas por el gobierno alemán para evitarla, para ver si han resultado efectivas dichas medidas. Utilizando los datos del ministerio de sanidad alemán, se trata de inferir los parámetros epidemiológicos centrales y el momento y magnitud de los efectos de la intervención gubernamental. En el desarrollo del artículo se especifica detalladamente toda la mecánica bayesiana a fin de crear el modelo PyMC3 para el estudio de este caso: qué tipo de aproximación de la distribución *a posteriori* se usó, qué algoritmo se utilizó para hacer la inferencia de las muestras por MCMC, los tipos de distribuciones *a priori* elegidas, etc. En conclusión, se determina que las futuras intervenciones del Estado alemán, sobre todo las de levantamiento de medidas restrictivas, se deben aplicar con precaución para respetar la visibilidad retardada de sus efectos porque se concluye que otras veces no han resultado ser efectivas a largo plazo.

[1] https://github.com/jonsedar/pymc3_vs_pystan

3 Diseño

3.1. Caso de estudio: blanqueo de capitales

Las entidades financieras como los bancos son sujetos obligados de la ley de prevención de blanqueo de capitales, lo que les exige entre otras cosas el monitorizar a sus clientes para alertar a las autoridades en caso de que su comportamiento sea sospechoso. Esta monitorización se realiza generalmente tanto en base a desviaciones de la operativa esperada, como en base a mostrar patrones o indicios que son consistentes de una actividad de blanqueo; y los casos suficientemente relevantes desatan una investigación interna. Por intentar modelar un comportamiento estadísticamente, un sistema de monitorización transaccional de blanqueo de capitales es un caso de estudio interesante para el uso de técnicas bayesianas.

Existe multitud de bibliografía sobre casos de fraudes monetarios encontrados en entidades financieras [20]. De este informe se seleccionó uno de entre todos los casos, el número catorce, para tener un ejemplo de referencia a implementar. Este suceso trata de un delito de lavado de dinero con beneficio ilícito proveniente de actividades ilegales de pornografía en Bielorrusia.

Los indicadores más relevantes en los que se basaron las instituciones judiciales para determinar que se estaba delante de un caso de blanqueo de capitales fueron las siguientes:

- El uso constante de criptomonedas por el beneficiario.
- La existencia de distintas cuentas en múltiples jurisdicciones, así como de estructuras empresariales fuera del territorio nacional.
- Retiradas de dinero en efectivo en cajeros ajenos a la empresa y retiradas de dinero constante en criptomonedas.
- La propiedad de vehículos de alta gama sin justificación clara de cómo se obtenían.

3.2. Creación de la Base de Datos de entrada

A falta de una base de datos (BBDD) que refleje la realidad exacta de la cuestión a tratar, se ha desarrollado una base de datos 'sintética' que es lo más fiel al caso de estudio que se desea modelar. Esta BBDD cuenta, entre otras cualidades, que es amplia y que refleja la realidad del tema tratado con objetividad, variedad, claridad y minuciosidad. En la elaboración de esta base de datos se han tenido en cuenta, además del ejemplo expuesto en el punto anterior, diferentes aspectos logísticos y económicos de los bancos y las transacciones de las cuentas bancarias, además de considerar cómo se está comportando y transformando el mundo financiero en nuestros días. Para ello se han realizado diferentes análisis, a fin de conseguir las mejores variables para luego modelarlas en PyMC3. Se han estudiado artículos sobre los datos bancarios necesarios para realizar transferencias entre entidades [21]; el significado y las diferencias entre los bancos digitales vanguardistas y la banca tradicional [22]; los indicadores más relevantes asociados al blanqueo de capitales [23] y un artículo científico donde se exponen técnicas de *machine learning* para detectar

transacciones ilegales [24], entre otras muchas cosas. Con todo ello, se ha llegado a las siguientes conclusiones que definen a alto nivel la base de datos detallada:

- La base de datos se compone de 100000 transacciones bancarias individuales, las cuales están etiquetadas por diferentes parámetros/variables. Intenta simular lo que puede ser una cuenta bancaria real dentro de una entidad financiera.
- Cada transacción está clasificada de manera booleana dependiendo si se considera que es *criminal* o *normal*: cero para comportamiento *normal* y uno para comportamiento *criminal*. Se define como *criminal* al tipo de conducta de una persona que comete un delito de blanqueo de capitales.
- Los parámetros que componen y dan información sobre la transacción están razonados sobre casos reales de cuentas bancarias, intentando especificar los más importantes para llegar a identificar, posteriormente con PyMC3, si una cuenta es *normal* o *criminal*.
- Estos parámetros no son homogéneos ni tienen todos el mismo peso dentro de la cuenta bancaria. Además, se calculan todos ellos de manera aleatoria siguiendo diferentes distribuciones de probabilidad. Se pretende dar así una imagen de variabilidad e incertidumbre en la cuenta bancaria, reflejando así el mundo real.

Para definir los parámetros y explicar la lógica seguida en el proceso de creación de la base de datos, se han agrupado y clasificado todos ellos en cinco grupos diferentes dependiendo de cómo los bancos son capaces de conseguir y recopilar los datos para determinar la información de la transacción. Además, se indica el tipo de dato que lo compone y en qué unidad se miden. Son los siguientes:

- Datos personales que posee el banco:
    - **Edad** del titular de la cuenta (Dato numérico decimal).
    - **Años de antigüedad** de la cuenta desde su creación (Dato numérico entero).
    - ¿**Dispone** el titular **de un crédito** con la entidad bancaria? (Booleano: Si/No).
    - ¿La transacción ha hecho que la **cuenta** quede **en negativo**? (Booleano: Si/No).
- Datos transaccionales que posee el banco de manera directa:
    - **Tipo de transacción** (Dato categórico: Nacional/Internacional).
    - **Dia** en el mes en que se realizó la transacción (Dato numérico entero: Del día uno hasta el día treinta).
    - **Gasto específico** de cada transacción (Dato numérico decimal).
- Datos transaccionales que posee el banco de manera indirecta:
    - **Concepto** propio de cada transacción (Dato categórico: diez tipos en total reflejados en la BBDD: compras en tiendas/supermercados, gastos en gasolinera, retiros en el cajero, transferencias bancarias, facturas (de todo tipo, cuenta también suscripciones), nómina, alquiler de la vivienda,

inversiones financieras, compras de materiales de lujo y, por último, otros conceptos).

- **Veces** en que se realiza al mes la transacción según el concepto (Dato numérico entero).

- Datos compartidos entre los bancos:
  - **Tipo de banco involucrado** en la transacción (Dato categórico: Banca Privada o Fintech/Neobanco/Banco Digital).
  - **Cantidad de cuentas** bancarias abiertas que posee el titular **en España** (Dato numérico entero).

- Otros tipos de datos:
  - **Nivel de actividad** de la cuenta (Dato categórico: Alto/Medio/Bajo).

Por otra parte, un matiz para tener en cuenta es que la generación de los datos dentro de la BBDD se ha dividido en dos principales bloques con el objetivo de diferenciar las transacciones en *normales* y *criminales*. En la *Figura 3-1* y en la *Figura 3-2* se puede apreciar esta división, además de la ayuda en la organización de la creación de la BBDD.

Concepto de la Transacción ID	Peso total	Peso total normal	Peso total criminal	DIA normal	DIA Criminal	VECES normal	VECES criminal	GASTO individual normal	GASTO INDIVIDUAL criminal	Tipo Transacción (Nacional / Internacional) normal	Tipo Transacción (Nacional / Internacional) criminal
Store purchases 0	22%	27.7%	11.76%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Poisson (3.5)	Poisson (4.0)	Truncnormal (mu=46, sd=50, L=1, H=120)	Truncnormal (mu=46, sd=50, L=1, H=120)	100 – 0	100 – 0
Gas Station 1	20%	19.69%	20.58%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Binomial (n=7, p=0.23)	Binomial (n=9, p=0.23)	Truncnormal (mu=40, sd=10, L=1, H=70)	Truncnormal (mu=40, sd=10, L=1, H=70)	100 – 0	100 – 0
Withdrawal 2	10%	8.63%	12.64%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Poisson (2.0)	Poisson (3.0)	Truncnormal (mu=40, sd=50, L=20, H=140)	Truncnormal (mu=55, sd=65, L=20, H=200)	95 – 5	80 – 20
Bank Transfers 3	10%	9.84%	10.29%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Binomial (n=5, p=0.4)	Binomial (n=7, p=0.45)	Truncnormal (mu=150, sd=60, L=50, H=500)	Truncnormal (mu=220, sd=80, L=50, H=1000)	90 – 10	65 – 35
Invoices 4	15%	11.36%	22.05%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Poisson (1.8)	Poisson (2.8)	Truncnormal (mu=40, sd=25, L=0, H=120)	Truncnormal (mu=40, sd=25, L=0, H=120)	95 – 5	85 – 15
Salary 5	5%	5.3%	4.41%	Betabinomial (n=30, a=6.0, b=1.2)	Betabinomial (n=30, a=3.9, b=1.3)	Bernoulli (0.8)	Bernoulli (0.75)	Truncnormal (mu=1500, sd=600, L=900, H=2400)	Truncnormal (mu=2000, sd=1000, L=900, H=3200)	100 – 0	100 – 0
Housing rent 6	3%	2.7%	3.53%	Betabinomial (n=30, a=1.2, b=7.5)	Betabinomial (n=30, a=1.2, b=4.0)	Bernoulli (0.75)	Poisson (1.1)	Truncnormal (mu=400, sd=100, L=250, H=470)	Truncnormal (mu=450, sd=100, L=250 H=600)	100 – 0	100 – 0
Financial investments 7	3%	2.27%	4.41%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Binomial (n=5, p=0.2)	Binomial (n=6, p=0.4)	Truncnormal (mu=200, sd=150, L=50, H=500)	Truncnormal (mu=350, sd=150, L=100, H=650)	50 – 50	20 – 80
Luxury materials purchases 8	2%	0.76%	4.41%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Poisson (1.0)	Poisson (2.0)	Truncnormal (mu=2500, sd=1300, L=1000, H=10000)	Truncnormal (mu=5000, sd=1400, L=1000, H=10000)	90 – 10	80 – 20
Others concepts 9	10%	12.12%	5.88%	Discrete Uniform (L=0, H=30)	Discrete Uniform (L=0, H=30)	Binomial (n=10, p=0.3)	Binomial (n=10, p=0.3)	Truncnormal (mu=42, sd=30, L=10, H=800)	Truncnormal (mu=42, sd=35, L=10, H=800)	80 – 20	75 – 25

Figura 3-1: Esquema de creación de la BBDD, bloque 1. Elaboración propia.

Name feature	Description
NEGATIVE ACCOUNT EVENT	Bernoulli (0.02) NORMAL || Bernoulli (0.04) CRIMINAL
YEARS ACCOUNT AGE	Betabinomial (n=8, a=0.8, b=0.6) NORMAL || Betabinomial (n=8, a=0.6, b=1.) CRIMINAL
ACCOUNT ACTIVITY LEVEL	Binomial (n=2, p=0.5) NORMAL-NACIONAL || Discrete Uniform (l=0, h=2) CRIMINAL-NACIONAL Binomial (n=2, p=0.67) NORMAL-INTER. || Binomial (n=2, p=0.75) CRIMINAL-INTER.
CREDIT PROVISION	Bernoulli (0.1) NORMAL || Bernoulli (0.02) CRIMINAL
TYPE OF BANK INVOLVED	Bernoulli (0.1) NORMAL-NACIONAL || Bernoulli (0.2) CRIMINAL-NACIONAL Bernoulli (0.5) NORMAL-INTER. || Bernoulli (0.8) CRIMINAL-INTER.
AGE OF ACCOUNT HOLDER	Truncnormal (mu=50, sd=26, l=18, h=92) NORM || Truncnormal (mu=38, sd=20, l=18, h=92) CRI
NUMBER ACCOUNTS HELD IN SPAIN	NegativeBinomial (n=2, p=0.78) NORMAL || NegativeBinomial (n=2, p=0.65) CRIMINAL

Figura 3-2: Esquema de creación de la BBDD, bloque 2. Elaboración propia.

Se puede observar a su vez un importante detalle a la hora de realizar el diseño, que indica la heterogeneidad de este. La tres primeras columnas de la *Figura 3.1*, que son: *Peso total, Peso total normal* y *Peso total criminal* se refieren al reparto de los diferentes conceptos a lo largo de la BBDD. Por ejemplo, sabiendo que el total de transferencias definidas en la BBDD es de 100000, y que el *Peso total* para el concepto de compras en tiendas es del 22%, indicaría que existen 22000 entradas de este concepto en la BBDD. Además de ese porcentaje total, existe un peso relativo especifico a la cuenta *normal* y *criminal*. De esta manera, se da un carácter verídico a la BBDD, pues es más probable ver conceptos de facturas en una cuenta *normal* que de compras de materiales de lujo o de gasolinera en una cuenta *criminal* que de compras en tiendas, por ejemplo.
Una vez conseguido el análisis de todas las variables presentes, solo queda rellenarlas de contenido 'simulado'. Como se puede ver tanto en *Figura 3-1* como en *Figura 3-2* todos esos parámetros están implementados en la base de datos de una manera aleatoria siguiendo diferentes distribuciones de probabilidad, las cuales se detallan a continuación justificando el porqué de su uso dependiendo de la variable en cuestión:

- Para la generación de la variable **día** de *Figura 3-1* se ha utilizado una distribución discreta uniforme (desde cero hasta treinta), salvo en los conceptos **alquiler** y **nómina** en los que hay diferencia entre *normal* y *criminal.* Para estos casos se ha usado una distribución Beta Binomial, que es la más adecuada para reflejar que los conceptos de **nómina** son más propensos en cuentas *normales* de ser a final de mes en comparación con las cuentas *criminales*. Lo mismo para el caso del concepto de **alquiler***,* pero esta vez siendo más propenso a que en cuentas *normales* se haga esta transacción a principios de mes en comparación con cuentas *criminales*, las cuales están más repartidas a lo largo del mes. En la *Figura 3-3* se puede ver cómo se han calculado los parámetros de la distribución Beta Binomial para esta variable.
- Para la generación de la variable **veces** de *Figura 3-1* se han usado tres tipos de distribuciones: Poisson, Binomial y Bernoulli. Todas ellas de carácter discreto, pues la variable *veces* solo puede registrarse como número entero. Para todos los conceptos hay diferencias en este caso, tanto entre unos y otros, como entre *normal* y *criminal*, con la excepción de la categoría de *otros conceptos* en la que es igual. Un ejemplo puede ser en el concepto de *inversiones financieras* en la que para la cuenta *normal* se define una distribución Binomial con parámetros $(n = 5, p = 0.2)$ y para la *criminal* los parámetros son $(n = 6, p = 0.4)$, remarcando así que es mucho más probable encontrarse conceptos de *inversiones* en los que la cuenta *criminal* es más propensa a hacerse con más frecuencia en el mes.

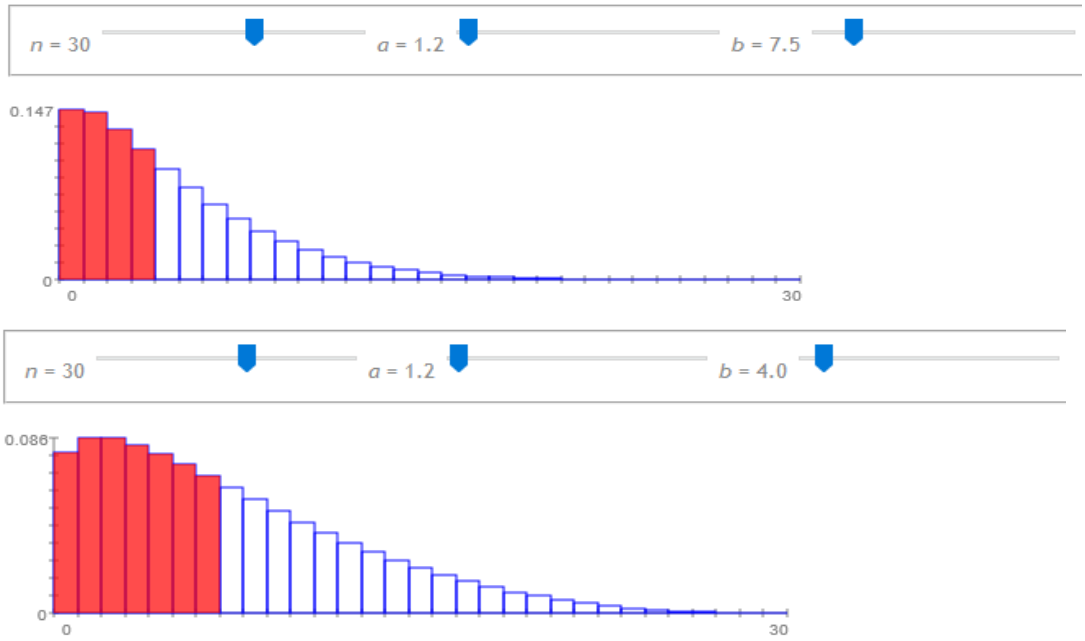


Figura 3-3: Arriba, para el caso de alquiler normal según el día del mes de la transacción. Abajo, para el caso de cuenta criminal. Extraído de [2].

- Para la generación de la variable **gasto específico** de *Figura 3-1* se ha empleado una única distribución continua: la distribución Normal-Truncada. Esta distribución de probabilidad es ideal porque permite acotar entre dos valores (Truncamiento) que se quiera, una distribución Normal con sus parámetros de media (μ) y desviación estándar (σ). Por poner un ejemplo, se puede analizar el caso para el concepto de *Transferencias Bancarias*. Para el caso de cuentas *normales*, se define una Normal-Truncada entre los valores de 50 € y 500 €, con media en 150 y desviación estándar en 60. Por otro lado, para las cuentas *criminales*, los valores de truncamiento son 50 € y 1000 €, con media en 220 y desviación estándar en 80. Así se expone como es más probable que para cuentas bancarias *criminales* los gastos sean mayores que para las *normales.*
- Para el caso del **tipo de transacción** de *Figura 3-1* se ha utilizado también una única distribución: Bernoulli. Puesto que es una variable categórica entre *nacional* o *internacional*, se puede convertir en variable binaria, es decir, indicando cero para transferencia de tipo *nacional* y uno para tipo *internacional*. Además, Bernoulli solo devuelve ceros o unos, e indicándole un parámetro *p* se le hace saber la probabilidad de que aparezca un cero o un uno. Por ejemplo, para el caso concreto de concreto de retiradas en un cajero, se define para la categoría *normal* una *p=0.95* y para *criminal, p=0.8*. Así se indica que es más probable encontrar unos para cuentas *criminales,* debido a que estos hacen más retiradas en cajeros internacionales que la gente con cuentas *normales*. Hay conceptos en los cuales todas las categorías son de tipo *nacional*, pues no se realizan intercambios de dinero con otros países a la hora de realizar la transacción, es el caso de: **compras en tiendas**, en **gasolineras**, **nómina** y **alquiler**. En definitiva, en la BBDD completa hay un 91,08% de posibilidades de

[2] http://www.distributome.org/js/calc/BetaBinomialCalculator.html

que se encuentre con una transferencia de tipo *nacional*, y un 8,92% de tipo *internacional*.

- Pasando al otro bloque de generación de los datos (como se puede ver en *Figura 3-2*), existen siete tipos de variables distintas:
    - En el caso de **evento de cuenta en negativo** se ha empleado Bernoulli, pues también se está ante una variable binaria. Se intenta plasmar como es más probable para las cuentas *criminales* quedarse en negativo que para las cuentas *normales*.
    - Para la **antigüedad de la cuenta** se ha empleado una distribución Beta Binomial que llega hasta ocho años. Se limita hasta ocho el número máximo de años para no sobrecargar al modelo, por lo que en este caso, cuando se pida a la distribución Binomial valores aleatorios de ella y muestre un ocho, será referido a ocho o más años de antigüedad de la cuenta. Se refleja cómo para el caso de cuenta *normal* es más probable tener cuentas más antiguas que para el caso *criminal.*
    - Con respecto a la variable **nivel de actividad de la cuenta**, en vez de repartirse diferentes parámetros de las distribuciones como pasa en otras variables para diferencia *normal* y *criminal*, en este caso se crean dos subgrupos dentro de ellos dependiendo el tipo de transacción (*Nacional* o *Internacional*), debido a que el nivel de uso de la cuenta va a depender también de la frecuencia de uso de transacciones dentro o fuera del país. Hay que remarcar también que debido a que hay tres tipos de actividad descritos en la BBDD (Alto, Medio y Bajo), se indexan las categorías para identificar los niveles (dos para el más alto, uno para el nivel medio y cero para el más bajo). Se usan las distribuciones Binomial y Uniforme Discreta, con la idea de remarcar lo reflejado en la *Figura 3-4*.

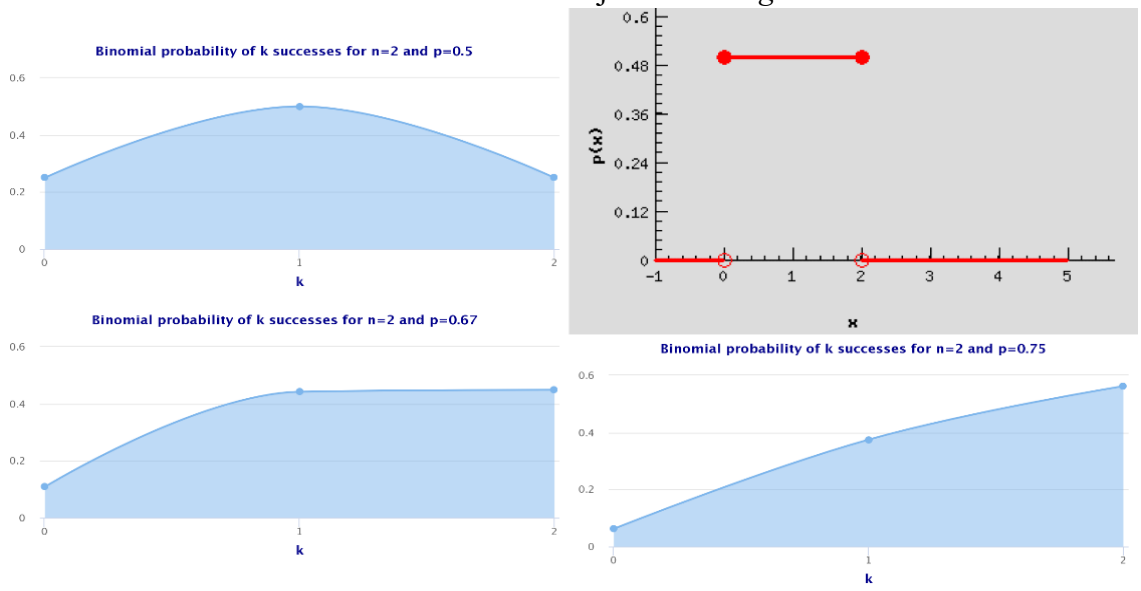


Figura 3-4: Parámetros utilizados para la distribución Binomial en relación con la variable nivel de actividad de la cuenta. Extraído de [34].

[3] https://www.anesi.com/binomial.htm?p=0.75&n=2
[4] http://www.elektro-energetika.cz/calculations/ro.php

- En relación con la variable de **tipo de banco involucrado** también se analiza con la misma estructura que la variable anterior, pero esta vez solo se usa la distribución Bernoulli para los cuatro casos. Al ser una variable categórica de dos únicas opciones (o es de tipo banca privada/tradicional o es de tipo fintech/neobanco/banca digital) se convierte en variable binaria. Se trata de dar más cantidad de unos (que refiere a tipo fintech/neobanco/banca digital) en los tipos *internacionales*, y dentro de ella, a las cuentas *criminales*.

- Para la variable de **provisión de crédito** se usa también solo Bernoulli, pues es los datos son booleanos (Si/No). Se busca reflejar como es más probable que una cuenta *normal* tenga un crédito con el banco que una cuenta *criminal.*

- En relación con la **edad del titular**, se pasa la variable de discreta a continua, con el objetivo de utilizar una distribución Truncada-Normal, acotada entre los 18 años (edad mínima en España para ser titular de una cuenta) y los 92 años. Los parámetros μ y σ que determinan la Normal dentro de estos valores están definidos de tal forma que se evidencie cómo gente más joven es más propensa a ser titular de una cuenta *criminal.*

- En último lugar, con respecto a la variable *número de cuentas en España del titular* se emplea una distribución Negativa-Binomial. La justificación de usar esta distribución y no la tradicional Binomial o la Beta Binomial, es porque en estas últimas no existe un parámetro *n* que limite el número de posibles valores de salida. En la *Figura 3-5* se muestra las diferentes funciones de densidad de probabilidad (PDF) para este caso en cuestión.

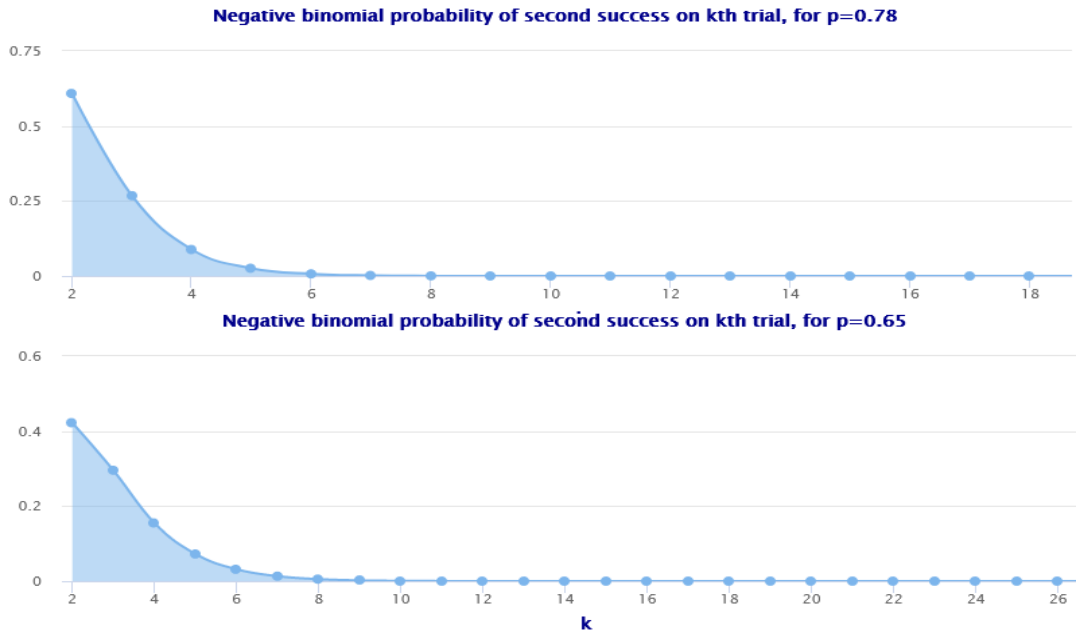


Figura 3-5: Parámetros utilizados para la distribución Negativa-Binomial. Extraído de [5].

[5] https://anesi.com/negativebinomial.htm?p=0.65&r=2

Por último, cabe recalcar que hay una relación total de 66%-34% de transacciones etiquetadas en la BBDD como *normales* y *criminales,* respectivamente (es decir, 66000 *normales* – 34000 *normales*). Para generar toda esta BBDD en Python se han empleado los métodos de generación de números aleatorios de las librerías matemáticas tanto de *NumPy*[6] como de *Seaborn*[7], las cuales contienen multitud de distribuciones de probabilidad diferentes para escoger. Por otra parte, a la hora de obtener los parámetros más idóneos en cada caso se ha usado la ayuda de una herramienta web[8][9], la cual está referenciada en los pies de las figuras. En ella se ha definido la elaboración de las formas y curvas de las PDF de cada distribución que más se ajustaban a nuestros criterios.

3.3. La transformación previa de los datos

A veces es necesario realizar una transformación de los datos previa a la fase de modelar. Se hace para que el modelo en cuestión converja de tal manera que los resultados posteriores no se vean alterados debido a que existan, por ejemplo, variables con órdenes de magnitud grandes en comparación con otras más pequeñas. Esto influye a la hora de '*samplear*' nuestro modelo, una de las fases más importantes en la hoja de ruta de PyMC3, la cual se explicará más adelante.

En nuestro caso, se tienen tres variables a las que es necesario transformar de algún modo:

- La primera de ellas es la variable **día**. Dado que existen diferencias en nuestra BBDD dependiendo del día del mes en que se ejecutó la transacción, se han agrupado en cinco bloques dependiendo si el día se encuentra a principio, a mediados, o a finales de mes. Se realiza a través de una función sencilla en Python.
- La siguiente es la variable **gasto específico**. Esta es la variable que posee más valores dispares en ordenes de magnitud de todas las de la BBDD, debido a que, tras lo especificado a la hora de realizar la BBDD, puede haber una transacción con un gasto de 10000 € (dentro del concepto de compras de materiales de lujo) como de 1 € (dentro del concepto de compras en tiendas/supermercado). Por ello, tras varias pruebas para ver cuál podía ser la transformación más acorde a los datos de **gasto especifico**, se ha decidido aplicarles el logaritmo decimal. En la *Figura 3-6* se puede ver cómo la transformación de los datos hace que se acumulen en torno a una franja, y no están tan dispersos.

[6] https://numpy.org/doc/1.16/reference/routines.random.html
[7] https://docs.scipy.org/doc/scipy/reference/stats.html
[8] http://www.distributome.org/js/calc/
[9] https://anesi.com/

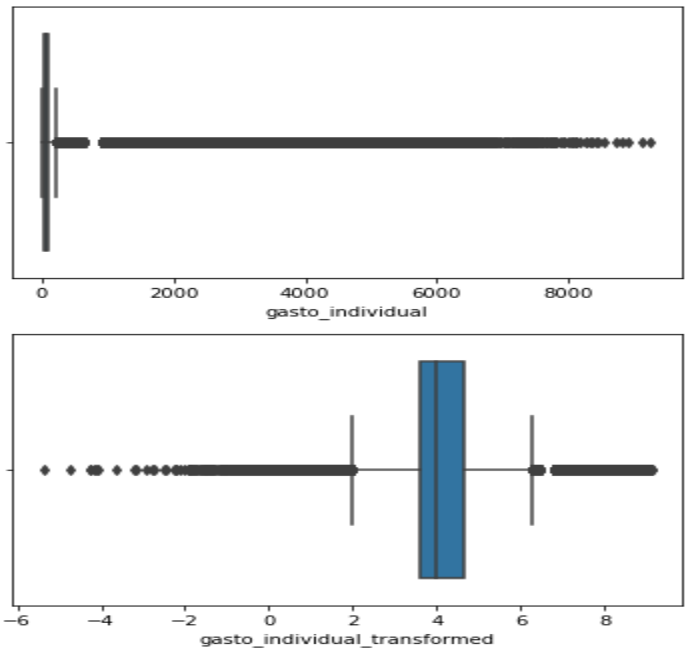


Figura 3-6: Arriba, diagrama de caja de la variable gasto especifico sin transformar. Abajo, diagrama de caja con los datos ya transformados. Elaboración propia.

- La última es la variable de *edad del titular*. La elección de esta transformación de esta variable, en la que se realiza el cuadrado de la división entre cuarenta de los datos de entrada, se ha apoyado en un caso de ejemplo de regresión logística en PyMC3 de la documentación oficial del framework [25], en la cual se realiza una transformación de este tipo en una variable similar, y, además, se prueba y justifica que los efectos en el modelo son positivos.

Por último, hay que señalar que en gran parte de la elaboración del programa se intentó realizar una transformación del tipo *one-hot encoding*, por la cual las variables categóricas sin relación lineal entre ellas se convierten dividiéndose en columnas en una forma binaria que proporciona a los algoritmos de *Machine Learning* un mejor trabajo de predicción. Esta codificación se intentó introducir en la variable **tipo de concepto**, la cual tiene 10 tipos diferentes, resultando ineficiente para el modelo, además de inútil para los resultados posteriores, por ello se desechó esta opción.

Se utilizará, en consecuencia, las siguientes variables (trece en total) de nuestra BBDD para la fase de modelar con PyMC3: **día agrupado**, **veces**, **gasto especifico transformado**, **cuenta en negativo**, **antigüedad de la cuenta**, **edad transformada**, **número de cuentas en España**, **tipo de banco involucrado**, **disposición de crédito**, **nivel de actividad de la cuenta, tipo de concepto**, **tipo de transacción** y la clasificación en *normal* o *criminal.*

3.4. El análisis exploratorio de la BBDD

En este apartado se va a examinar de manera analítica y estadística la naturaleza de nuestra BBDD 'sintética'. Este paso se realiza en todo proyecto relativo a datos, para verificar que no ha habido ningún error en la ingesta de ellos, observar además algunas características y para resaltar matices estadísticos, como la posible correlación entre las variables. Las

siguientes figuras se elaboran mediante la librería de Python seaborn, una de las mejores bibliotecas de visualización de datos en Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

Observando la *Figura 3-6* se pueden sacar valiosas conclusiones acerca de la BBDD de entrada. Por una parte, las variables más reseñables en cuanto a un claro patrón de diferencia entre cuenta *normal* o *criminal* pueden ser la de **antigüedad de la cuenta**, la cual se aprecia que existen más cuentas *criminales* nuevas que *normales.* Otra variable puede ser la de edad, donde se aprecia que la curva para *criminal* indica que con hay más probabilidad de encontrar con una persona joven disponiendo de una cuenta *criminal* que siendo más viejo. Es muy importante representar gráficamente las distintas variables del modelo para ver con qué tipo de comportamientos de los datos se va a enfrentar; en nuestro particular caso, no sorprenden ciertos resultados pues la BBDD fue de elaboración propia, pero en la gran mayoría de casos no se tienen conocimientos previos.

En la *Figura 3-7* se observa cómo no se aprecian correlaciones muy fuertes con la variable clasificación, nuestra variable objetivo del modelo. La correlación más alta positiva es 0.27 correspondiente a antigüedad de la cuenta, por lo que no es necesario descartar ninguna variable para el modelo. En versiones anteriores del programa se vieron correlaciones muy altas entre las variables, lo que hizo reescribir la BBDD sintética, encontrando fallos y puliendo los parámetros de las distribuciones incluidas.

En este apartado también es esencial saber con qué tipo de datos se va a trabajar. En la *Figura 3-8* se puede observar que de todas las variables de nuestra BBDD (sin transformar) ocho son categóricas, por lo que necesitarán algún tipo de transformación para convertirlas en numéricas; cinco son de tipo datos enteros y una única variable es de orden decimal.

También es útil en esta parte exponer ‘rankings’ donde se muestren por ejemplo las 10 edades más presentes en el BBDD o representar gráficamente más funciones de seaborn como las scatterplot (o también llamados gráficos de dispersión en español), la cual ayuda a encontrar relaciones entre dos variables, útil para comprobar si puede existir una relación lineal o cuadrática, identificar patrones o clúster en los datos, etc.

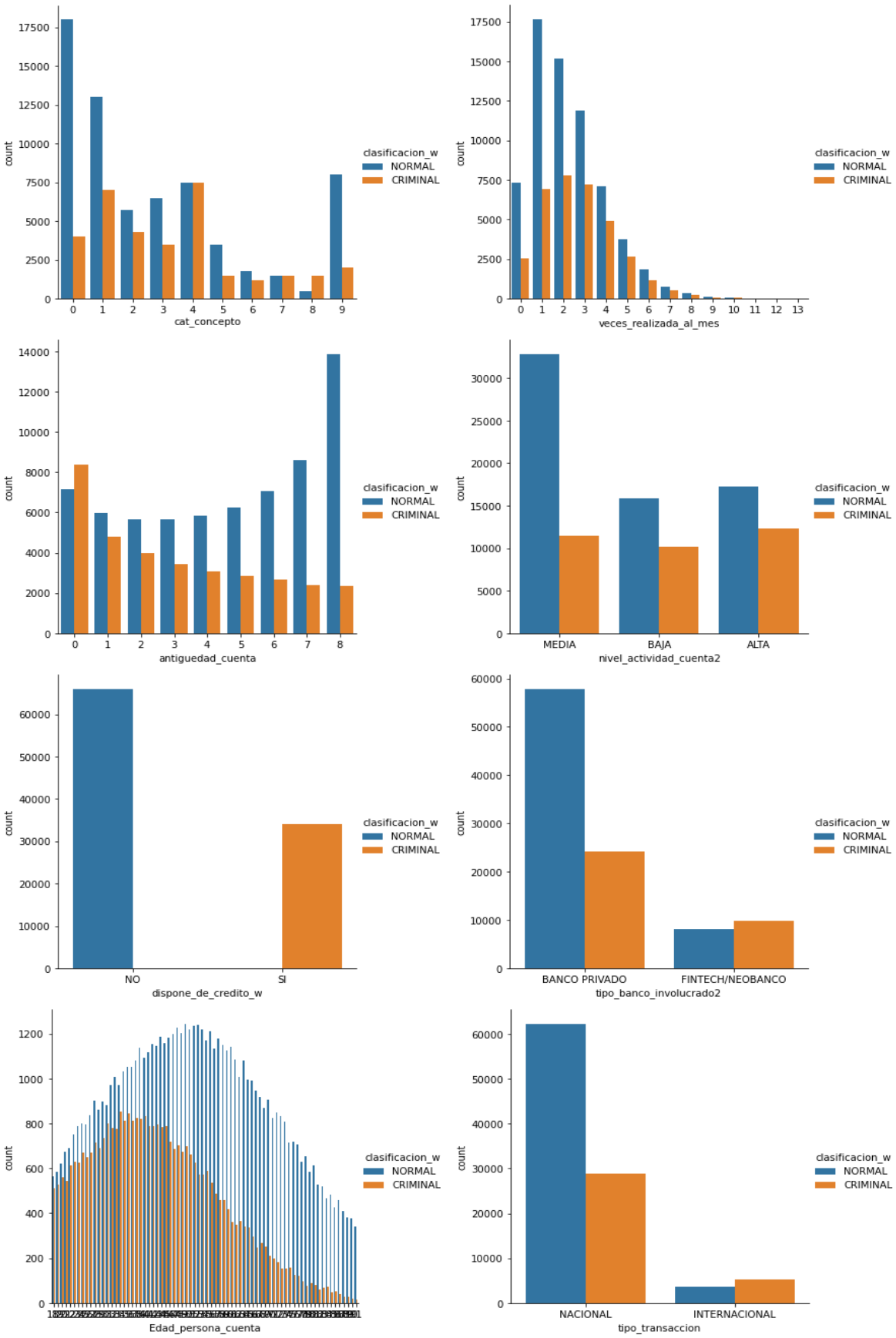


Figura 3-6: Gráficas de barras donde se comparan diferentes variables en función de su clasificación en normales o criminales. Elaboración propia.

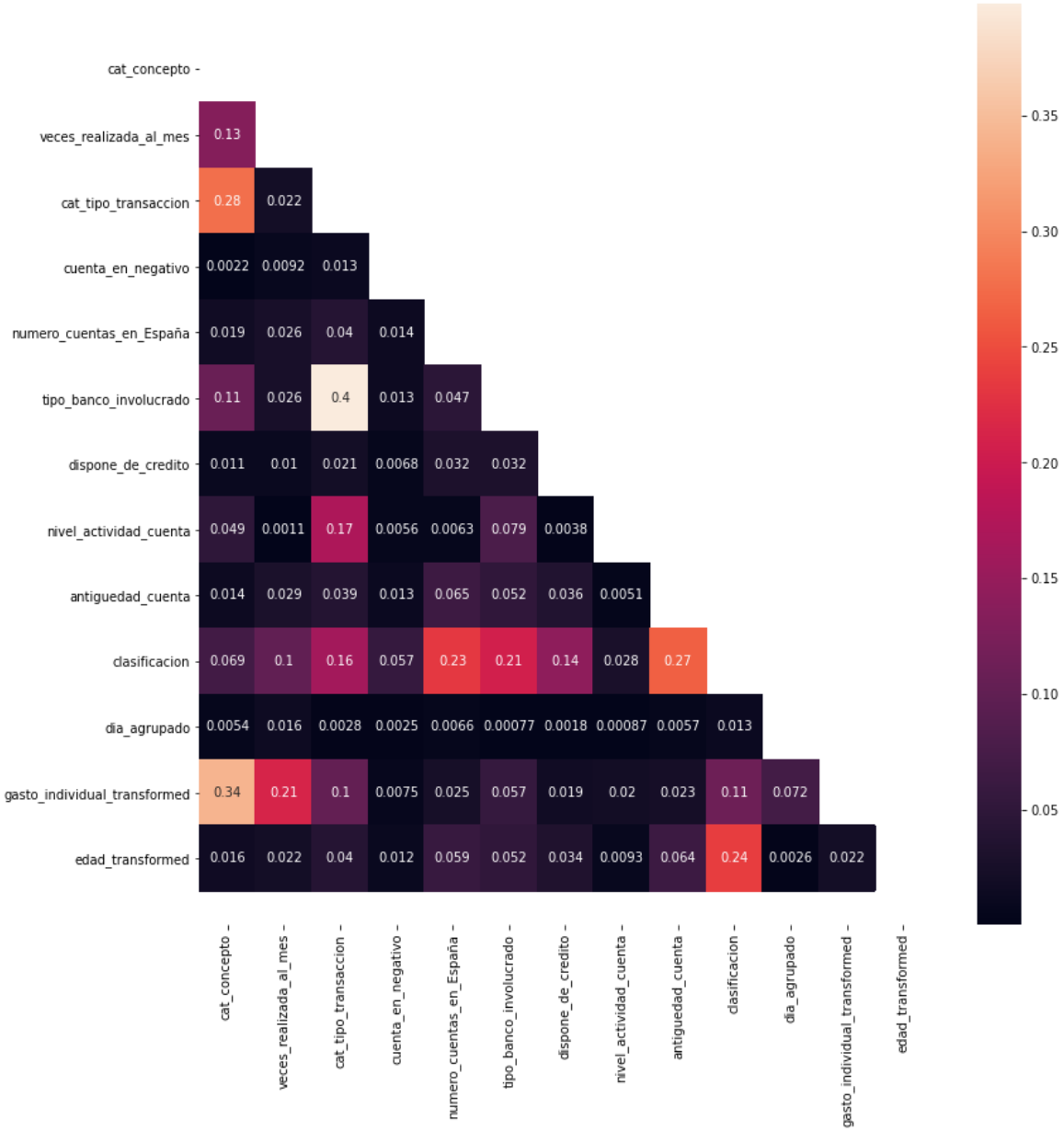


Figura 3-7: Matriz en forma de mapa de calor donde se exponen las correlaciones entre las distintas variables que se utilizan en el modelo. Elaboración propia.

```
dia                          int32
veces_realizada_al_mes       int32
gasto_individual           float64
numero_cuentas_en_España     int32
Edad_persona_cuenta          int32
antiguedad_cuenta            int32
concepto_transaccion        object
tipo_transaccion            object
tipo_banco_involucrado2     object
nivel_actividad_cuenta2     object
clasificacion_w             object
dispone_de_credito_w        object
cuenta_en_negativo_w        object
```

Figura 3-8: Tipos de datos de las variables (sin transformar). Elaboración propia.

4 Desarrollo

4.1. Construyendo el modelo en PyMC3

Estos datos de la BBDD creada previamente se usarán para modelarlos de tal manera que se puedan sacar conclusiones usando programación probabilística, y en concreto utilizando el framework PyMC3. Se tiene muy presente el Teorema de Bayes en todo momento (ya se especificó su definición en la *Figura 2-1*), por lo que para entender bien la sintaxis del modelo, hay que estar familiarizado con la lógica bayesiana. El objetivo principal es crear un **modelo PyMC3 de regresión logística que sea jerárquico y centrado**.

Figura 4-1: Implementación del modelo PyMC3 en relación con el caso de estudio, identificando con un número las diferentes partes que lo componen. Elaboración propia.

Como se puede ver en la *Figura 4-1,* lo primero que se crea es un objeto en Python llamado *Model*, que encapsula todas las variables que componen nuestro modelo teórico, manteniéndolas en un único contenedor para que puedan ser utilizadas como una única unidad posteriormente. Se puede observar también que el modelo fue declarado utilizando una expresión `with`. Esta expresión se utiliza para definir un lenguaje de Python conocido como gestor o administrador de contexto, que permite gestionar al usuario diferentes tipos de recursos de algún tipo de programa, en este caso, de nuestro modelo PyMC3. Ayuda principalmente a automatizar la introducción de las variables definidas en el modelo, así cualquier variable que se escriba debajo de esta sentencia, se añade directamente al modelo,

sin tener que añadirla explícitamente. Esto evita la sintaxis repetitiva de diferentes métodos relacionados con el objeto que se está especificando. Una vez creado el Modelo, se puede rellenarlo con todos los componentes que se haya especificado en el estudio de nuestro caso.

La estructura del modelo se puede dividir en cinco partes las cuales se han identificado en la *Figura 4-1* para facilitar la explicación:

La **primera** fase es donde se crean las llamadas 'coordenadas del modelo', las cuales ayudan a identificar las gráficas de las distribuciones *a posteriori* que resultan después de la inferencia bayesiana. Como estas distribuciones *a posteriori* están organizadas en *arrays*, etiquetar las coordenadas y las dimensiones de estos con sus nombres exactos facilitan el análisis posterior. Esta utilidad se integró en PyMC3 en la versión 3.9.0., y se integra a la perfección con la librería *ArviZ* que se utilizará más adelante para mostrar los resultados. Además, dentro de esta estructura también se usan dos contenedores de datos, definidas como *pm.Data*, que permiten al modelo ser consciente de sus entradas y salidas, además de otras ventajas como a la hora de generar predicciones posteriores del modelo. Como se puede ver en la *Figura 4-2*, estas dos variables están referidas al **tipo de concepto** de transacción y al **tipo de transacción**, que van a ser importantes porque van a catalogar en *arrays* ciertas distribuciones *a priori* de nuestro modelo.

```
cat = np.array(data_modelo['cat_tipo_transaccion'])
y_data = np.array(data_modelo['clasificacion'])
cat_concepto = np.array(data_modelo['cat_concepto'])
```

```
_, concepts = pd.factorize(data_pandas["concepto_transaccion"])
_, types_transaction = pd.factorize(data_pandas["tipo_transaccion"], sort=False)
coords = {"concepto": concepts, "tipo_transaccion": types_transaction, "total": np.arange(100000)}
```

Figura 4-2: Implementación, previa al modelo, de las variables para definir las coordenadas del modelo. Elaboración propia.

En la **segunda** parte de la estructura se definen las variables que definen nuestro modelo como **jerárquico**, y además que tiene un carácter **centrado**. Es decir, las variables que se especifican en este apartado se reutilizan más tarde en la parte tercera, que es donde se definen las distribuciones *a priori*. Esta forma de jerarquizar la sintaxis del modelo tiene una explicación: los modelos jerárquicos se utilizan porque una medida proporciona información para otra medida, aunque no estén relacionadas de forma causal. Por ejemplo, saber que cinco casos de la variable X tienen una media de 5 y una desviación estándar de 2, puede informar sobre los valores esperados de las siguientes observaciones de X, aunque no se tenga ninguna información concreta sobre cómo se relacionan los casos de X entre sí. Por el contrario, los modelos no jerárquicos parten de 'cero' al hacer inferencias sobre cada instancia de X. Esto significa que se necesitan más datos para reducir la incertidumbre en la misma medida que podría hacerlo un modelo jerárquico.

Por otro lado, escribir de manera centrada el modelo ayuda a separar los pesos de las variables de entrada (diez en total) al peso de la variable *intercept* de la ecuación de regresión logística, la cual no proporciona prácticamente ninguna información valiosa sobre los resultados posteriores (solo es esencial al mirar la convergencia del modelo), mientras las otras dan toda la información. Se puede observar estas diferencias en la *Figura 4-1*.

El esquema del modelo es jerárquico porque existe una capa por encima de todas las variables del modelo, y es centrado porque estas variables 'jerarcas' (llamadas así porque son los argumentos de las distribuciones *a priori,* es decir, la media y la desviación estándar)

separan la variable *intercept* de las demás. Estas variables jerarcas tendrán a su vez importancia en la elección de la distribución *a priori*, referente a la tercera parte de la estructura. En el Anexo se dibuja el esquema de nuestro caso.

En la **tercera** parte es donde se especifican todas las distribuciones *a priori* del modelo. Uno de los requisitos de la estadística bayesiana es que después de observar los datos, que se utilizarán en la línea donde se define la *likelihood* (la quinta parte), todas las demás variables son tratadas como variables aleatorias. Las variables aleatorias bayesianas no tienen por qué surgir necesariamente de un proceso aleatorio físico. Pero la interpretación que se le den a estas variables *a priori* va a depender del conocimiento previo que se tenga sobre ellas, modificando el tipo de distribución de probabilidad que se le va a asignar en el modelo. PyMC3 incluye la mayoría de las distribuciones más comunes usadas para el modelado estadístico.

Si las distribuciones *a priori* (P(θ) en el Teorema de Bayes) están mal elegidas, el algoritmo MCMC puede no converger, o al menos tener dificultades para hacerlo, lo que afectaría a nuestro modelo PyMC3. Por esta razón, es mejor elegir cuidadosamente las distribuciones *a priori*. A menudo, la falta de convergencia o la evidencia de que las muestras se amontonan en los límites implica que algo está mal con las distribuciones *a priori* elegidas.

El estudio de la elección de las distribuciones *a priori* a escoger para nuestro particular modelo, se ha apoyado en un artículo [26] donde se detalla de manera minuciosa las características dependiendo del conocimiento previo que se tenga sobre las variables y su entorno. Se usa la llamada distribución *a priori* 'débilmente informativa'; $Normal\ (\mu = 0,\ \sigma = 10)$, representa que se tiene una idea vaga de los posibles pesos de las variables que van a ir dentro del Teorema de Bayes, dilucidando que van a pertenecer al rango [-10,10], que es un rango muy amplio. Además, también representa que no se quiere cerrar la posibilidad de acortar el rango, pues se podría perder información valiosa que desvirtuaría el modelo. Es una elección a medio camino entre no tener ningún conocimiento previo, y tener claro cómo se puede comportar las variables posteriores.

Además, como se comenta también en el artículo, cuando se dice que esta prioridad es 'débilmente informativa', lo que se quiere decir es que, si hay una cantidad razonablemente grande de datos, la probabilidad dominará, y la distribución *a priori* que se escoja no será relevante. Sin embargo, si los datos son escasos, esta 'prioridad débilmente informativa' influirá mucho en la inferencia posterior. En nuestro caso particular, se tiene más de 100000 transacciones en la BBDD, cantidad muy grande que permitirá, escogiendo este tipo de distribuciones *a priori*, no preocuparse de primeras por la definición que se les haya dado a éstas en el modelo.

En la **cuarta** y **quinta** parte se puede ver justificado por qué nuestro modelo es del tipo **regresión logística.** La explicación teórica del uso de este tipo es porque nuestros datos observados, correspondientes a la columna (que es la variable objetivo) *clasificación* (como se puede observar en la *Figura 4-2*), son de tipo 'booleano', es decir, solo contiene ceros o unos (0 para el comportamiento *normal*, 1 para *criminal*). La regresión logística es perfecta para este tipo de casos en los que se necesita un algoritmo de clasificación de variables binarias discretas. La regresión logística transforma su resultado utilizando la función sigmoide, la cual se encarga de mapear cualquier valor real en otro valor entre cero y uno. En *machine learning*, se utiliza la sigmoide para mapear las predicciones en probabilidades. Es necesaria usarla en el modelo, pues si no se mapean estos valores, la *likelihood* devolvería

un error, ya que en la distribución Bernoulli no se puede introducir en el parámetro *p* valores más allá del rango [0,1]. La definición matemática de la función sigmoide está ilustrada en la *Figura 4-3.*

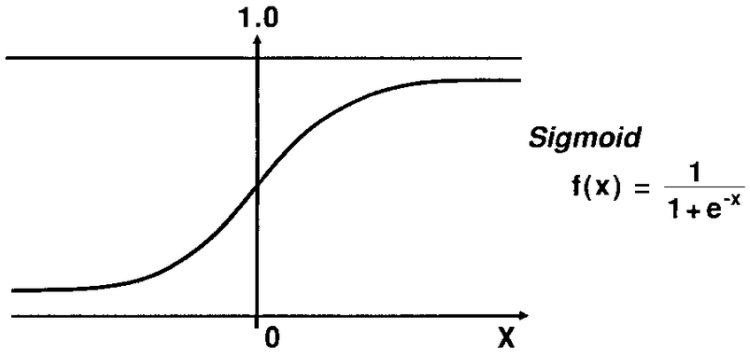


Figura 4-3: Representación gráfica de la función sigmoide. Extraído de: [27].

Como se ha comentado, la distribución elegida para la *likelihood* es la de Bernoulli (se puede ver en la parte 5 de la *Figura 4-1*), ya que se asume que la variable objetivo sigue la naturaleza de una variable aleatoria Bernoulli.

En consecuencia, se asume que la probabilidad de ser *criminal* o *normal* viene definida por una función de los **días agrupados**, **veces**, **gasto especifico** (estos tres últimos se ramifican a su vez en los **tipos de concepto**), **cuenta en negativo**, **años de antigüedad de la cuenta**, **edad del titular**, **cuentas del titular en España**, **disposición de crédito**, **tipo de banco involucrado** y **nivel de actividad de la cuenta** (estas dos últimas se ramifican a su vez en **tipo de transacción**). La fórmula de interpretación del problema se puede ver definida en la parte 4 de la *Figura 4-1* y ve reflejada en anotación matemática en la *Figura 4-5*.

$$
\begin{aligned}
p = Sigmoid(\ & \beta 0 + \beta 1(dias_{agrupados})[concepto] + \beta 2(\text{veces})[concepto] \\
& + \beta 3(\text{gasto específico}_{\text{transformado}})[concepto] + \beta 4(\text{cuenta en negativo}) \\
& + \beta 5(\text{antigüedad de la cuenta}) + \beta 6(\text{edad}_{\text{transformado}}) \\
& + \beta 7(\text{número de cuentas en España}) \\
& + \beta 8(\text{tipo de banco involucrado})[tipo\ transacción] \\
& + \beta 9(\text{dispone de crédito}) \\
& + \beta 10(\text{nivel de actividad de la cuenta})[tipo\ transacción]\ ) \\
& and\ y = 1\ if\ account\ is\ criminal\ and\ y = 0\ otherwise.
\end{aligned}
$$

Figura 4-5: Fórmula de la regresión logística para nuestro caso. Elaboración propia.

Por lo tanto, una vez especificadas tanto de manera jerárquica y centrada las distribuciones *a priori*, como una probabilidad *a posteriori* para extraer muestras, referida a la *likelihood*, además de dejar indicada la matemática del modelo, con todo ello resuelto se podrá dar por cerrada la parte de creación del modelo PyMC3.

4.2. Realización de la inferencia bayesiana

La función de muestreo de PyMC3 (*pm.sample*) ajusta los modelos probabilísticos como el especificado anteriormente, utilizando el muestreador de Monte Carlo de la cadena de

Markov (MCMC). En este paso es donde ocurre toda la inferencia bayesiana, con todos los cálculos que conlleva, por lo que es la fase en donde se gastan más recursos del ordenador. El tiempo aproximado de ejecución de nuestro modelo implementado está alrededor de 4 horas. A menos que se asigne manualmente algoritmos particulares a las variables de nuestro modelo, PyMC3 asignará por defecto los algoritmos que considere más apropiados (se ha probado que en esto el algoritmo de PyMC3 hace un trabajo decente en este sentido, por lo que se dejará en manos de PyMC3 esta decisión). Como se puede ver en la *Figura 4-6*, la definición del paso del muestreo va debajo de la instancia de Python `with`, como pasaba en la definición de la sintaxis del modelo, indicando así al que se va a referir. Además, se puede observar que se asigna por defecto el algoritmo NUTS.

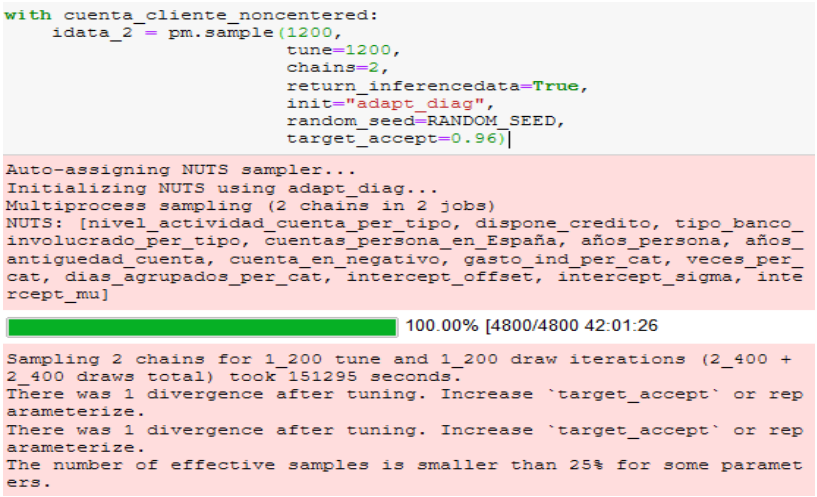


Figura 4-6: Sintaxis y resultados del proceso de muestreo del modelo. Elaboración propia.

Esta función devuelve las muestras de la cadena de Markov del modelo en una estructura de datos llamada *trace* (o traza, en español). Esta estructura será un objeto de Python de la librería contigua a PyMC3, *ArviZ*, en la cual se apoyará para exponer y presentar los análisis *a posteriori* del modelo de una manera sencilla e intuitiva. Asimismo, esta función siempre toma al menos un argumento, el número de *samples* (o muestras, en español) a dibujar. Después de una serie de pruebas probando distintos argumentos que lleva dentro nuestra función de muestreo, se definieron los siguientes [28]:

- Número de muestras a dibujar: 1200.
- `chain` = 2. El número de cadenas a muestrear. Ejecutar cadenas independientes es importante para algunas estadísticas de convergencia.
- `tune` = 1200. Por definición, es el número de iteraciones a afinar. Los muestreadores ajustan los tamaños del paso durante el tuning.
- `return_inferencedata = True`. Devuelve una trace como un objeto arviz.InferenceData, útil para nuestros análisis posteriores.
- `init='adapt_diag'`. El método de inicialización para usar en el algoritmo autoasignado NUTS.
- `random_seed=RANDOM_SEED`. Semilla para el generador de números aleatorios.
- `target_accepted=0.95`. Este argumento adapta el tamaño del paso de NUTS de forma que la probabilidad media de adaptación en todas las trayectorias del algoritmo se acerque al target_accepted. Ajustar a valores más altos ayuda con el

muestreo de distribuciones *a posteriori* más difíciles o que se 'atascan' más. Aunque también valores más altos hacen que el tiempo en muestrear suba considerablemente. Finalmente, tras probar con otros valores, se ajusta a 0.95.

4.3. Comprobación del modelo

Existen diferentes herramientas dentro del universo PyMC3 para valorar si un modelo está correctamente escrito, si los datos están introducidos de manera correcta, si converge o no, o en definitiva, comprobar si el algoritmo MCMC elegido está funcionando correctamente.

La primera de ellas es calcular, una vez escrita la sintaxis del modelo y antes de proceder a muestrear, la log-probabilidad de las distintas variables involucradas para ver que no haya ninguna incongruencia. Se computa mediante un método de PyMC3 que permite hacerlo de manera sencilla. Si cualquiera de ellas diese un valor $-\infty$, el modelo quedaria inmediatamente invalidado. En nuestro caso no se produce este hecho.

Una herramienta adicional es el método de *ArviZ*, *az.plot_energy,* el cual traza la distribución de transición de energía y distribución de energía marginal en los algoritmos HMC. Esto puede ayudar a diagnosticar una mala exploración por parte de los algoritmos basados en el gradiente, como HMC o NUTS. Como nuestro algoritmo es de tipo NUTS, se puede usar. Como se puede observar en la *Figura 4-7,* ambas distribuciones son idénticas, lo que indica que el modelo no se atasca dentro del algoritmo.

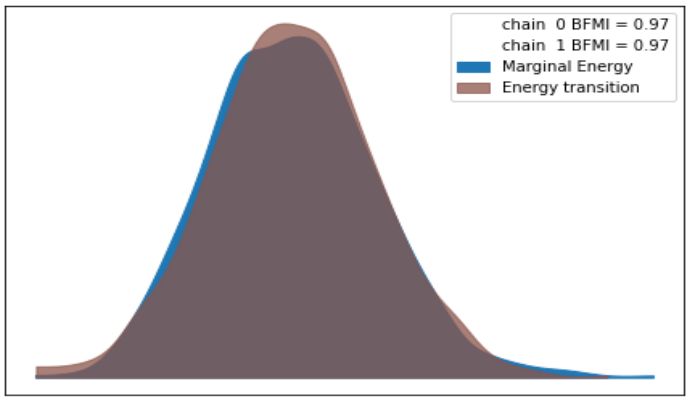


Figura 4-7: Gráfica de salida del método ArviZ az.plot_energy. Elaboración propia.

Otra herramienta útil es fijarse en tres parámetros que devuelve la *trace* para todas las variables una vez muestreado el modelo. Estos parámetros estadísticos son: *mcse_mean*, *mcse_sd* y *r_hat*. Estos parámetros nos dan una idea de la convergencia del modelo. Si los dos primeros están por debajo de 0.1 y el tercero está en el rango [1.00, 1.02] se puede dar luz verde al modelo. En nuestro modelo se cumple esta premisa. La última herramienta utilizada es verificar que nuestro modelo devuelve los atributos correctos. Para ello, se usa otro método de la biblioteca *ArviZ*, az.hdi. Como se puede observar en la *Figura 4-8*, el procedimiento en este caso es ver si la media de los dos valores de cada tupla de la imagen de la izquierda es igual al valor que se muestra en la misma posición del *array* en la imagen de la derecha. En nuestro modelo, esta norma se cumple.

```
trace_hdi = az.hdi(idata)
trace_hdi['gasto_ind']
```

```
<xarray.DataArray 'gasto_ind' (concepto: 10, hdi: 2)>
array([[0.09362565, 0.15400238],
       [0.33468898, 0.403191  ],
       [0.28426943, 0.34899533],
       [0.00561528, 0.06536184],
       [0.33446581, 0.39916364],
       [0.37117394, 0.4289531 ],
       [0.49965254, 0.58609738],
       [0.00918367, 0.07923233],
       [0.31659498, 0.36601403],
       [0.2034753 , 0.28312182]])
Coordinates:
  * concepto  (concepto) object 'STORE PURCHASE' 'GAS STATION' ... '
OTHERS'
  * hdi       (hdi) <U6 'lower' 'higher'
```

```
idata.posterior['gasto_ind'].median(('chain', 'draw'))
```

```
<xarray.DataArray 'gasto_ind' (concepto: 10)>
array([0.12497389, 0.36772264, 0.31580075, 0.03721055, 0.3644099 ,
       0.39895534, 0.54086388, 0.0449675 , 0.33894053, 0.24279751])
Coordinates:
  * concepto  (concepto) object 'STORE PURCHASE' 'GAS STATION' ... '
OTHERS'
```

Figura 4-8: Comprobación en el código del modelo para ver si se devuelven los valores correctos. Elaboración propia.

Como estas cuatro comprobaciones del modelo dan resultados positivos, se puede concluir que el modelo es apto.

4.4. Comparación con otros tipos de arquitecturas de modelos

Para comprobar y comparar la efectividad de nuestro modelo, se prueba el mismo caso de estudio pero con dos tipos de sintaxis PyMC3 diferentes, una jerárquica pero no centrada y otra no jerarquizada. La principal diferencia con el modelo que se ha explicado hasta ahora (el que es jerárquico centrado) es que en el primero de los dos se crean variables 'jerarcas' para cada variable 'hijo', y en el segundo no existen variables 'jerarcas', los parámetros se pasan directamente a las distribuciones *a priori*.

Para evaluar el rendimiento del modelo de regresión logística, se procede a usar las métricas de *F1-score* y *accuracy.* Los resultados se indican en la *Figura 4-9*.

```
Accuracy of the hirarchical-centered model: 0.77892
f1 score of the hirarchical-centered model: 0.635470254583828
Accuracy of the hirarchical-non centered model: 0.74352
f1 score of the hirarchical-non centered model: 0.618734067321474
Accuracy of the non-hirarchical model: 0.72699
f1 score of the non-hirarchical model: 0.602876320261117
```

Figura 4-9: Métricas para medir el comportamiento de los modelos. Elaboración propia.

Como puede verse, el que mejor resultado da es el modelo jerárquico centrado, resultando hasta un 77,89% de casos que el modelo ha acertado en la predicción. Estas métricas son muy usadas en todo proyecto en el que se usen algoritmos de *machine learning*, aunque son recomendables que se usen junto a otras, puesto que algunas pueden llevar a engaño resultando que el modelo posea *underfitting* o *overfitting*.

Otra de las razones que motivó a escoger el modelo jerárquico centrado como el modelo central de pruebas es que en la fase del muestreo era la que tenía una duración y unas divergencias posteriores más comprensibles, puesto que el modelo no jerárquico tardaba menos en muestrear, pero tenía muchas más divergencias, y el modelo jerárquico no centrado tardaba mucho más (a relación de días) en muestrear aunque tuviese menos divergencias.

5 Integración, pruebas y resultados

5.1. Resultados posteriores

En este apartado se van a exponer las conclusiones que se obtienen al muestrear el modelo. Se estudia y se trabaja sobre las distribuciones *a posteriori*, las distribuciones resultantes del Teorema de Bayes (ver *Figura 2-1*), el centro del análisis bayesiano. Durante todo este proceso se ha utilizado *ArviZ*, una librería complementaria a PyMC3 escrita en Python, que usa el análisis exploratorio de datos para los modelos bayesianos. Incluye funciones para el análisis de la distribución *a posteriori*, el almacenamiento de datos, el diagnóstico de muestras, la comprobación de los modelos y la comparación de ellos. El objetivo es proporcionar herramientas para el diagnóstico y la visualización de la inferencia bayesiana en Python. Se comienza exponiendo en *Figura 5-1* todas las distribuciones *a posteriori* que resultan de la inferencia del modelo.

Se puede ver que todas ellas siguen un patrón en forma de distribución normal, lo que tiene sentido ya que se está pasando como distribuciones *a prori* al modelo solo distribuciones Normales (ver *Figura 4-1*). Igualmente, muchas de estas distribuciones resultantes se presentas más 'achatadas' que otras. La explicación de esto es la incertidumbre alrededor de esa variable, cuanta más incertidumbre, más 'achatada' estará su distribución *a posteriori*. Dos de las causas de este fenómeno puede ser la escasez de datos sobre esa variable de estudio, o que los datos se encuentran muy repartidos, a grandes escalas de magnitud, lo que sería un problema para la convergencia del modelo. Esta última posibilidad queda descartada al haber afinado nuestro modelo para que no ocurriese, realizando las transformaciones oportunas a los parámetros que lo necesitasen (explicado en la *Sección 3.4*). Asimismo, en la *Figura 5-1* se presenta la salida del método de *ArviZ*, *az.plot_trace*, el cual genera dos columnas, la de la izquierda es la que llama a *plot_dist* para trazar el KDE/Histograma de los datos (dependiendo si son distribuciones continuas (KDE) o discretas (Histograma), en nuestro caso al ser distribuciones normales es del primero), y la columna de la derecha contiene la propia traza, los valores de las muestras individuales por cada paso del algoritmo, con la longitud del número de samples a dibujar que se haya asignado en la parte del muestreo del modelo.

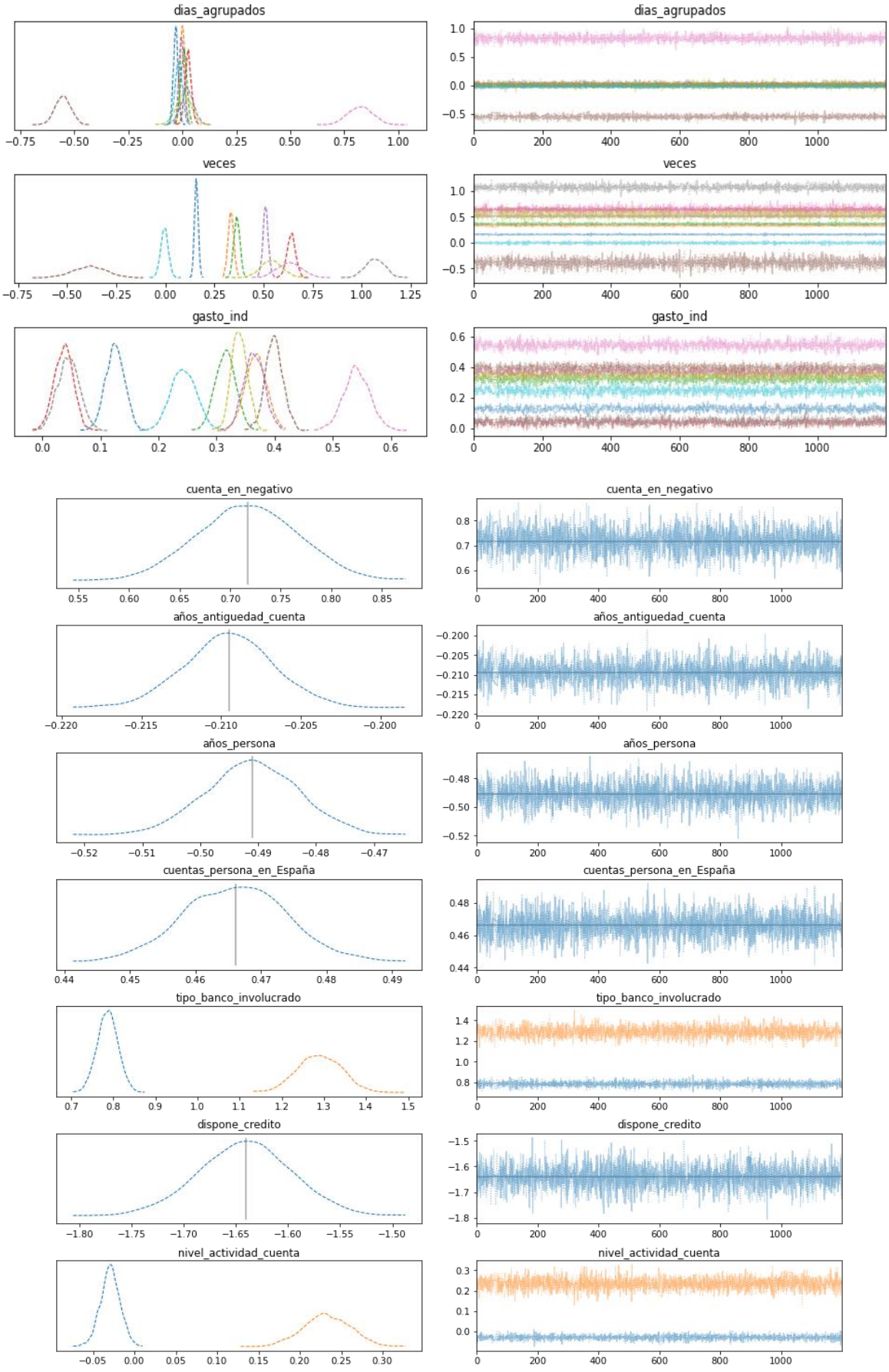


Figura 5-1: Distribuciones posteriores de nuestro modelo. Elaboración propia.

A continuación se van a mostrar en forma gráfica los pesos (en forma de *odds ratio*) de cada una de las variables de nuestro modelo de regresión logística, muy útiles para entender las características del problema que se está abordando:

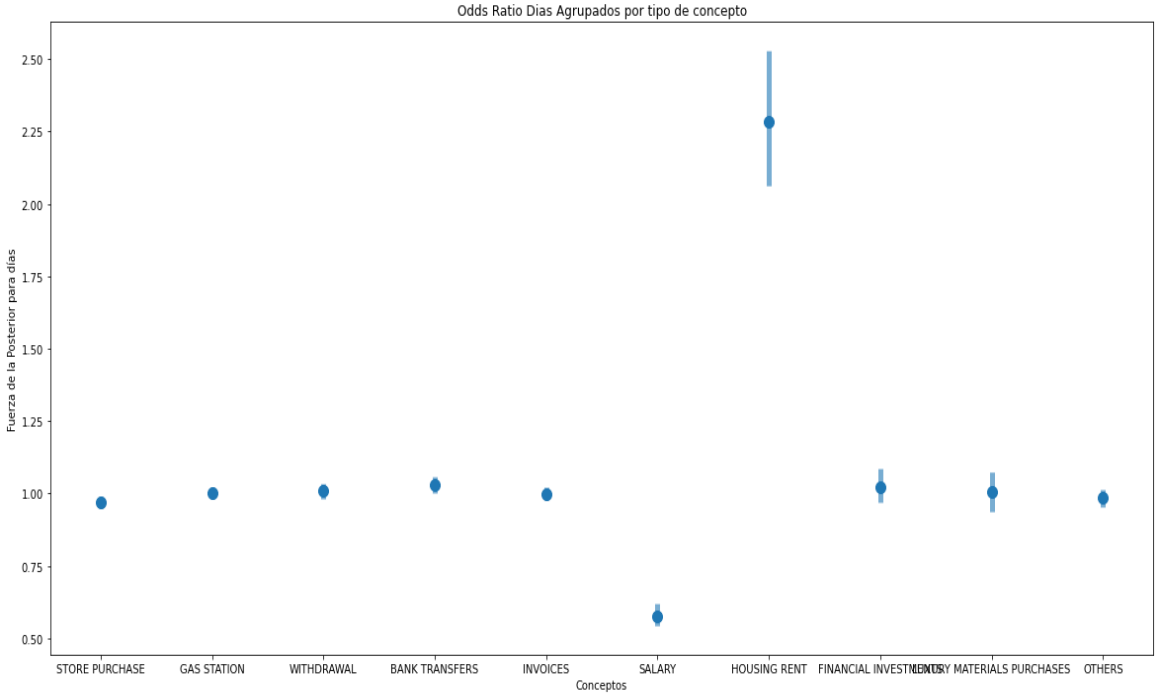


Figura 5-2: Odds Ratio correspondiente a la variable de día. Elaboración propia.

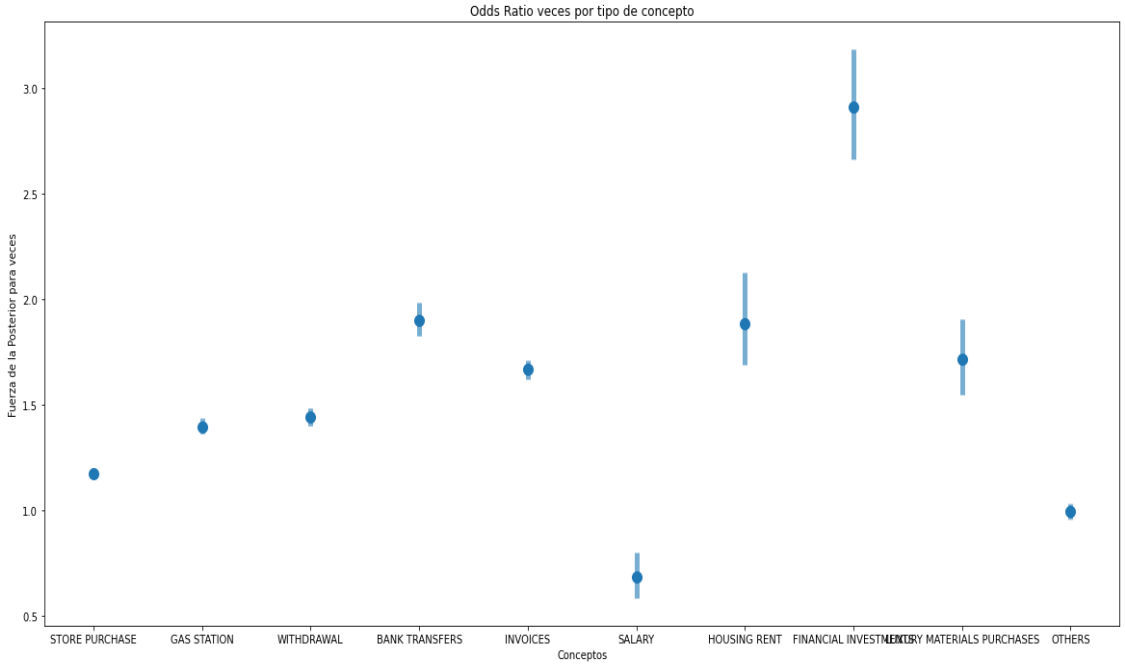


Figura 5-3: Odds ratio correspondiente a la variable veces. Elaboración propia.

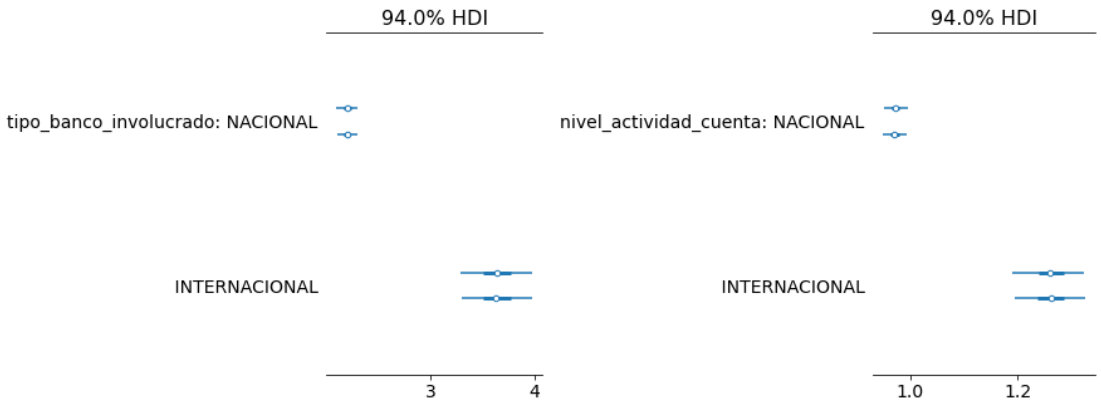


Figura 5-5: Odds ratio correspondiente a las variables tipo de banco involucrado (izquierda) y nivel de actividad de la cuenta (derecha). Elaboración propia.

Figura 5-6: Odds ratio correspondiente a las demás variables que conforman nuestro modelo. Elaboración propia.

La razón de usar el *'odds ratio'* es, en cierto modo, la 'escala' en la que viven las variables en una regresión logística. Mientras que en una regresión lineal, se puede interpretar un coeficiente de, por ejemplo, 1.5 como que por cada aumento de una unidad de la variable de predicción (por ejemplo, años del poseedor de una cuenta bancaria), hay un aumento de 1.5 en la variable de predicción (por ejemplo, los ingresos por nomina en una cuenta), en una regresión logística las cosas son un poco diferentes.

En este caso, se predicen resultados binarios (0-1, por ejemplo, cuenta correspondiente a *criminal* o *normal*) y un *coeficiente exponencial* de 1.5 indica que por cada aumento de una unidad de la variable de predicción, hay un aumento de 1.5 en las 'probabilidades' (es decir, la probabilidad relativa o p/(1-p)) de que la variable de predicción sea 1 (es decir, cuenta

criminal) en lugar de 0. Dicho de otro modo, indica que por cada año adicional en la edad del poseedor de la cuenta, hay 1.5 veces más probabilidades de ser *criminal* que de ser *normal*, en comparación con alguien con un año menos. Por ello, todos los pesos (o coeficientes) de las variables se les aplica una transformación exponencial para obtener las respectivas razones de probabilidad que implican. Esto se debe a que la regresión logística trabaja con el logaritmo de las probabilidades y no con las probabilidades directamente.

Se puede apreciar también que en todas las figuras de arriba existe un círculo correspondiente al valor medio, y aparte, en algunas se distingue una línea azul a ambos lados del valor medio. Una ventaja de la estadística bayesiana en comparación con la frecuentista es que se tiene mucha más información que un valor medio. En particular, se puede calcular el intervalo de alta densidad (HDI) del 94% para esos parámetros, la cual está reflejada en esa línea azul que se hacía referencia anteriormente. A continuación se exponen varias conclusiones de los resultados obtenidos:

- Observando la *Figura 5-2* se muestra los coeficientes de la variable **día** dividida por el **tipo de concepto**. La mayoría de los conceptos no inciden nada en el modelo (tienen peso ≅ 0), ya que la variable **día** fue calculada de manera uniforme en la BBDD. En cambio, para los conceptos de **nómina** y **alquiler** sí importa, y más en el último, el día en que se realiza la transacción.
- Con respecto a la *Figura 5-3*, esta vez sí existen muchas diferencias en los coeficientes de cada concepto. La explicación está en la manera en que se generaron estos datos, utilizando distribuciones de probabilidad diferentes, con parámetros diferentes.
- Se puede resaltar el ejemplo de un par de casos interesantes. Por una parte, en el concepto de **inversiones financieras** se observa como la *'odds ratio'* en **veces** es muy grande (con valor medio cerca de 3) pero en cambio en **gasto especifico** es insignificante para el modelo (está alrededor de 1), debido a que siguiendo los patrones de nuestro caso de estudio, los conceptos de **inversiones financieras** eran superiores en número pero no en cantidad para cuentas *criminales.* Otro concepto reseñable es el de **facturas***,* el cual se puede interpretar como un concepto con un valor muy seguro (pues no se aprecia línea de HDI 94%, ya que toda ella está dentro del círculo de valor medio) en torno a 1.5 para el parámetro de **veces***,* y un coeficiente parecido para **gasto individual**. Esto viene de la mano de que hay mismas probabilidades de que con cada aumento en **veces** y **gasto especifico** en transacciones de facturas, la cuenta resultante sea de tipo *criminal.*
- La variable con más poder de proporcionar información de que una cuenta es *criminal* es la de **tipo de banco involucrado**, como se puede ver en el gráfico de la izquierda de la *Figura 5-5*, con un coeficiente de hasta 4.0 para transacciones *internacional* y de 2.5 para transacciones *nacionales*. Esto explica que hay muchas posibilidades de encontrarse con una cuenta *criminal* cuando se usan bancos del tipo fintech/neobancos, pues son más útiles para intercambiar dinero de manera digital. Además, otra conclusión que se puede apreciar de las gráficas de la *Figura 5-5* es la fuerza en cuanto a probabilidades de ser *criminal* en transacciones que se hagan entre países fuera del territorio nacional.
- Por último, se muestran las demás gráficas de las variables que no dependen de otras en la *Figura 5-6.* Por reseñar alguna, la variable **disposición de crédito** se puede ver que posee coeficiente negativo, esto quiere decir que cuanto más sea la probabilidad de ver créditos financieros en la cuenta, menos probabilidades hay de que sea

criminal. Pasa de una forma muy parecida para las variables de la **edad del titular** y **años de antigüedad de la cuenta**.

- En las *Figuras 5-5* y *5-6* se pueden observar cómo se repite dos veces las marcas correspondientes a cada variable, esto es debido a que en el paso de muestrear el modelo se ha utilizado dos cadenas independientes, como si se repitiesen los procesos de muestreo dos veces. Como se ve, son idénticas todas ellas, no habiendo diferencias apreciables. Esto es una muestra más de que nuestro modelo converge de manera excelente.

El tercer paso en la evolución de los resultados de las distintas distribuciones *a posteriori* resultantes será la obtención de predicciones sobre el modelo. Se utiliza para ello el método de PyMC3, *pm.sample_posterior_predictive,* al que se le pasan como argumentos la trace que sale de la fase de muestreo y el nombre del modelo PyMC3, el cual queda definido en la primera línea al construir la sintaxis.

Para obtener estas predicciones, se evalúa el objeto creado con la sentencia anteriormente descrita en posiciones diferentes a las observaciones reales de nuestro modelo. En nuestro programa, se ha querido calcular unas métricas utilizando el valor medio de los parámetros predichos como una estimación 'más probable'. La solución ha sido aproximada a una matriz de confusión muy utilizada en la evaluación de algoritmos de *machine learning*. El resultado se puede encontrar en la *Figura 5-7*.

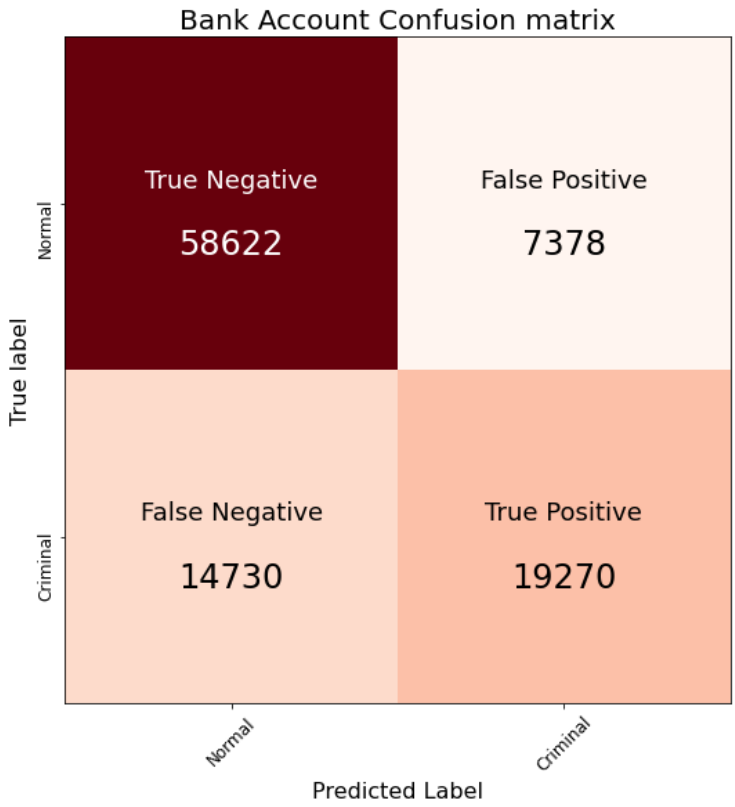


Figura 5-7: Matriz de confusión de las observaciones predichas por nuestro modelo. Elaboración propia.

5.2. Herramientas utilizadas

Las herramientas que se usan en este TFG se obtienen directamente de *Anaconda Navigator*, una distribución de carácter libre y abierta para la ciencia de datos y el *machine learning*, que contiene los lenguajes más conocidos dentro del aprendizaje automático como son R y Python. Posee una interfaz gráfica de usuario incluida en la instalación de la distribución. En *Anaconda* se pueden encontrar los programas más comunes para trabajar con esos ambientes, además existe la opción de usar la consola de línea de comandos *Anaconda prompt* desde la que se puede gestionar todo el material disponible. Los paquetes se administran y se despliegan mediante el llamado sistema de gestión de paquetes *conda*, una herramienta muy sencilla de usar y actualizar. Algunos ejemplos del software que se puede encontrar son: *JupyterLab*, *Spyder*, *RStudio* o *TensorFlow*.

La instalación de la distribución *Anaconda* se hace directamente desde su página web[10] eligiendo el programa de instalación que concuerde con nuestro tipo de Sistema Operativo. Se recomienda trabajar desde la *Anaconda Prompt* para gestionar todos los paquetes y acciones sobre las librerías que engloban *Anaconda*.

Dentro de *Anaconda* se encuentra la herramienta *jupyter notebook* que es muy útil para crear, compartir y desarrollar trabajos y proyectos relacionados en el lenguaje Python, aunque éste no es el único lenguaje compatible para *jupyter notebook*. Su faceta intuitiva para hacer anotaciones, documentar funcionalidades, insertar ecuaciones y visualizar resultados hacen que sea la herramienta elegida en este TFG para la realización del programa.

Para ejecutar PyMC3 es necesario disponer de un entorno compatible con Python, al menos en su versión 3.5. Se ha trabajado creando un entorno especifico en *Anaconda* con las librerías necesarias para correr el programa de este proyecto, ya que si no se hace ese entorno y se descargan las librerías por libre, pueden producirse errores con la compatibilidad de versiones entre paquetes en un futuro. PyMC3 se instala mediante la gestión de paquetes *conda* dentro de la distribución *Anaconda.* Además, como se ha comentado, también depende de terceros paquetes de Python, que se instalarán automáticamente al instalar PyMC3 a través de *conda*. Las librerías adicionales necesarias, independientemente de PyMC3, son: *Theano*, *NumPy*, *SciPy*, *Matplotlib* y *ArviZ*. Para sacar el máximo provecho a PyMC3 también es recomendable instalar *seaborn*, *pandas*, *sklearn*, *itertool*s, *IPython*, *pyjanitor* y *graphviz*, todas ellas empleadas en el proyecto.

[10] www.anaconda.com

6 Conclusiones y trabajo futuro

6.1. Conclusiones

El problema del lavado de dinero ha sido, es y será una de las mayores amenazas para los departamentos de prevención de fraude de los países de la UE, entre ellos España. Año tras año se dedican grandes cantidades de dinero desde los presupuestos generales de los Estados para poder reducir estas actividades ilícitas. Mediante este TFG se ha querido mostrar una posible ayuda para conseguir atajar este problema, empleando para ello la herramienta de programación probabilística PyMC3.

La manera en la que se ha intentado exponer PyMC3 ha sido de carácter amplio y general, es decir, exponiendo desde la base teórica de dónde se sustenta (Teorema de Bayes y MCMC), pasando por indicar su paquete back-end (*Theano*) como el funcionamiento del propio PyMC3, hasta presentar un ejemplo de uso en un caso específico de blanqueo de capitales.

Se ha conseguido mostrar en el desarrollo y los resultados del TFG cómo puede funcionar este framework simulando una cuenta bancaria con distintas transacciones, cuya BBDD ha sido diseñada de manera propia. Con ello, se ha llegado a recapitular información sobre las características de los parámetros y las variables de ella en función de la naturaleza de la cuenta. Además, gracias a los resultados se puede hacer una idea de si una cuenta bancaria tiene más probabilidades de ser una cuenta *normal* o ser una cuenta *criminal,* pudiendo llevar a la práctica la detección preventiva de blanqueo de capitales en cualquier entidad financiera.

6.2. Trabajo futuro

Partiendo del trabajo realizado y el conocimiento obtenido en el desarrollo de este TFG, se plantean diversos objetivos como trabajo futuro:

- Uno de los mayores aspectos por dónde poder progresar sería en el desarrollo de una BBDD más realista que la diseñada en este proyecto, partiendo de que en nuestra BBDD la relación de transacciones *normales*/*criminales* es de un 66-34 % respectivamente, y la lógica de la realidad nos indica que en un banco tradicional la relación es mucho menor.
- Siguiendo con el hilo del punto anterior, un asunto muy interesante podría ser la inclusión de muchas más características de una cuenta bancaria a los datos de entrada, con el objetivo de que puedan ayudar a ofrecer más información sobre si una cuenta está siendo utilizada de manera fraudulenta.
- Otro de los puntos a mejorar podría ser la búsqueda de otra arquitectura de modelo PyMC3, buscando tanto el rediseño de las distribuciones *a priori* como de la *likelihood*, que permita aumentar la efectividad en las predicciones en relación con el modelo presentado en este proyecto.
- También sería interesante poder exprimir más las capacidades de la librería en Python sobre visualización de datos de inferencia bayesiana, *ArviZ*, explicada en este proyecto, para así conseguir más y mejores visualizaciones que ayuden a entender los datos posteriores provenientes del muestreo del modelo PyMC3.

7 Referencias

[1] AML6: La sexta Directiva Europea en Blanqueo de Capitales, 22 Noviembre 2020. [En línea]. Available: https://www.electronicid.eu/es/blog/post/aml6-sexta-directiva-blanqueo-capitales/es. [Último acceso: 20 Mayo 2021].

[2] J. Valero, Europa quiere una autoridad central contra el lavado de dinero, *El Economista,* 30 Octubre 2020.

[3] Bruselas pide mejorar la lucha contra el blanqueo de capitales y la evasión fiscal de las 'golden visa', *El Pais,* 23 Enero 2019.

[4] C. Davidson-Pilon, Bayesian Methods For Hackers: Probabilistic Programming and Bayesian Methods, Pearson Addison Wesley, 2015.

[5] B. Cronin, What is probabilistic programming?, 16 Abril 2013. [En línea]. Available: https://www.oreilly.com/content/probabilistic-programming/. [Último acceso: 20 Mayo 2021].

[6] About PyMC3, 2018. [En línea]. Available: https://docs.pymc.io/about.html. [Último acceso: 20 Mayo 2021].

[7] R. E. L. Briega, Introducción a la inferencia Bayesiana con Python, 21 Mayo 2017. [En línea]. Available: https://relopezbriega.github.io/blog/2017/05/21/introduccion-a-la-inferencia-bayesiana-con-python/. [Último acceso: 20 Mayo 2021].

[8] J. J. Luna, Métodos MonteCarlo basados encadenas de Markov, 2015.

[9] Cadenas de Markov: Teoría y ejemplos, [En línea]. Available: http://investigaciondeoperaciones2markov.blogspot.com/p/teoria-y-ejemplos.html. [Último acceso: 19 Mayo 2021].

[10] J. Santaolalla, ¿En qué consiste el método Montecarlo?, 9 Julio 2015. [En línea]. Available: https://naukas.com/2015/07/09/video-en-que-consiste-el-metodo-montecarlo/. [Último acceso: 20 Mayo 2021].

[11] T. U. o. B. Columbia. [En línea]. Available: https://wiki.ubc.ca/Course:CPSC522/MCMC. [Último acceso: 20 Mayo 2021].

[12] What is EDA?, [En línea]. Available: https://www.itl.nist.gov/div898/handbook/eda/section1/eda11.htm. [Último acceso: 20 Mayo 2021].

[13] A. Gelman, A Bayesian Formulation of Exploratory Data Analysis and Goodness-of-fit Testing, 2003.

[14] PyMC3 and Theano, [En línea]. Available: https://docs.pymc.io/PyMC3_and_Theano.html. [Último acceso: 20 Mayo 2021].

[15] P. Developers, The Future of PyMC3, or: Theano is Dead, Long Live Theano, 26 Octubre 2020. [En línea]. Available: https://pymc-devs.medium.com/the-future-of-pymc3-or-theano-is-dead-long-live-theano-d8005f8a0e9b. [Último acceso: 19 Mayo 2021].

[16] A beginner's guide to Theano for Deep Learning, 5 Enero 2020. [En línea]. Available: https://recodeminds.com/blog/a-beginners-guide-to-theano-for-deep-learning/. [Último acceso: 20 Mayo 2021].

[17] P. Coyle, A Hierarchical model for Rugby prediction, [En línea]. Available: https://docs.pymc.io/notebooks/rugby_analytics.html. [Último acceso: 20 Mayo 2021].

[18] M. Á. Méndez, Cómo Volkswagen engañó a todos trucando sus coches con un 'software', *El Confidencial,* 22 Septiembre 2015.

[19] J. Z. J. S. F. P. W. M. N. J. P. W. M. &. P. V. Dehning, Inferring change points in the spread of COVID-19 reveals the effectiveness of interventions., *Science,* 2020.

[20] Financial Analysis Cases 2011 - 2013, Toronto, 2015.

[21] Datos bancarios necesarios para realizar transferencias entre entidades, 15 Octubre 2020. [En línea]. Available: https://ayudaleyprotecciondatos.es/2020/10/15/datos-bancarios-transferencias-entidades/. [Último acceso: 19 Mayo 2021].

[22] M. Olivé, Neobanco, Fintech, Challenger Bank, BaaS ¿Cuál es la diferencia?, 12 Mayo 2021. [En línea]. Available: https://financer.com/es/blog/neobanco-fintech-challengerbank/. [Último acceso: 20 Mayo 2021].

[23] 6 indicadores de riesgo asociados al blanqueo de capitales, 23 Agosto 2019. [En línea]. Available: https://www.ealde.es/riesgos-blanqueo-capitales/. [Último acceso: 19 Mayo 2021].

[24] M. L. A. H. R. Å. G. y. L. J. Jullum, Detecting money laundering transactions with machine learning, *Journal of Money Laundering Control,* vol. 23, nº 1, pp. 173-186, 2020.

[25] P. C. a. J. B. Cook, GLM: Logistic Regression, 15 Septiembre 2020. [En línea]. Available: https://docs.pymc.io/notebooks/GLM-logistic.html. [Último acceso: 19 Mayo 2021].

[26] A. Gelman, Prior Choice Recommendations, 18 Abril 2020. [En línea]. Available: https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations. [Último acceso: 20 Mayo 2021].

[27] T. Næs, Artificial Neural Networks in Multivariate Calibration, *Journal of Near Infrared Spectroscopy,* p. 2, 1993.

[28] API Reference: Inference, Sampling, [En línea]. Available: https://docs.pymc.io/api/inference.html. [Último acceso: 19 Mayo 2021].

8 Glosario

Accuracy: Es una forma de medir la frecuencia con la que un algoritmo de Machine Learning clasifica correctamente un conjunto de datos.

BBDD: Base de Datos

Blanqueo de capitales: es un delito que sanciona a todo aquel que, a sabiendas del origen ilícito de los bienes, realice actuaciones con la finalidad de lucrarse de ellos y poder sacar un beneficio extra.

CERN: Organización Europea para la Investigación Nuclear

EDA: Análisis exploratorio de datos

F1-score: Combinación de las medidas de precisión y exhaustividad para devolver una medida de calidad más general del modelo.

HDI: Highest Density Interval

HMC: Hamiltonian Monte Carlo

Ingeniería Fiscal: Es la utilización de instrumentos financieros para reestructurar un perfil financiero existente y obtener así otro con propiedades más deseables.

Machine Learning: Rama del campo de la Inteligencia Artificial que busca dotar a las máquinas de la capacidad de aprendizaje.

MCMC: Métodos de Monte Carlo basado en cadenas de Markov

NUTS: Muestreo de No U-Turn

Programación Probabilística (PP): La programación probabilística es un paradigma de programación en el que se especifican modelos probabilísticos, y la inferencia a partir de estos modelos se realiza automáticamente.

Overfitting: Concepto de *machine learning*, correspondiente al estado de un modelo cuando éste se aprende los datos de entrenamiento perfectamente, por lo que no es capaz de generalizar, y cuando le lleguen nuevos datos obtendrá pésimos resultados.

UE: Unión Europea

Underfitting: Concepto de *machine learning*, correspondiente al estado de un modelo cuando éste no es capaz de identificar patrones, por lo que tendrá siempre pésimos resultados.

9 Anexo

A Manual de Instalación y configuración

Para la instalación en Windows mediante Anaconda, se siguieron los siguientes pasos:

- `conda create -c msys2 -c conda-forge -n mypm3env python=3.8 mkl-service libpython m2w64-toolchain scipy matplotlib pandas`
- `conda activate mypm3env`
- `conda install -c conda-forge pymc3`

Si se tiene *Theano* originalmente instalado, hay que desinstalarlo e instalar *Theano-PyMC* a través de Conda Forge:

- `conda remove theano`
- `conda install -c conda-forge theano-pymc`

Otras dependencias instaladas para nuestro programa:

- `conda install -c conda-forge arviz`
- `conda install -c anaconda scikit-learn`
- `conda install -c anaconda seaborn`
- `conda install -c conda-forge r-itertools`
- `conda install -c conda-forge python-graphviz`

Todos los modelos PyMC3 han sido corridos por una computadora de las siguientes características:

- Procesador Intel Core i3-9100F CPU @ 3.60GHz, 3600 MHz, 4 núcleos.
- Memoria RAM de 16 GB
- Sistema Operativo Microsoft Windows 10 Pro
- Tarjeta Gráfica Nvidia GeForce GTX 1660 6GB GDDR5
- Disco Duro SSD 250 GB

B Problemas y dificultades descubiertas

A la hora de instalar PyMC3 en la prompt de Anaconda, hubo ciertos problemas para importar este paquete a jupyter notebook y que reconociese las líneas dónde era usado. Este problema era referido como un problema en la verificación de la versión del 'existing lazylinker compiled file'. El error que se produce se debe a la instalación de *Theano* de modo implícito con conda. En este caso lo que se debe hacer es: Desinstalar el paquete de la distribución, instalar en conda el paquete m2w64-toolchain que proporciona una versión totalmente compatible del compilador GCC con g++, éste último del que se pueden escribir

muchas líneas, pero básicamente es mejor que *Theano* recurra a un compilador C como este que permite una ejecución mucho más rápida que otros compiladores. Después se volvió a instalar *Theano* de manera independiente en conda y se actualizó PyMC3.

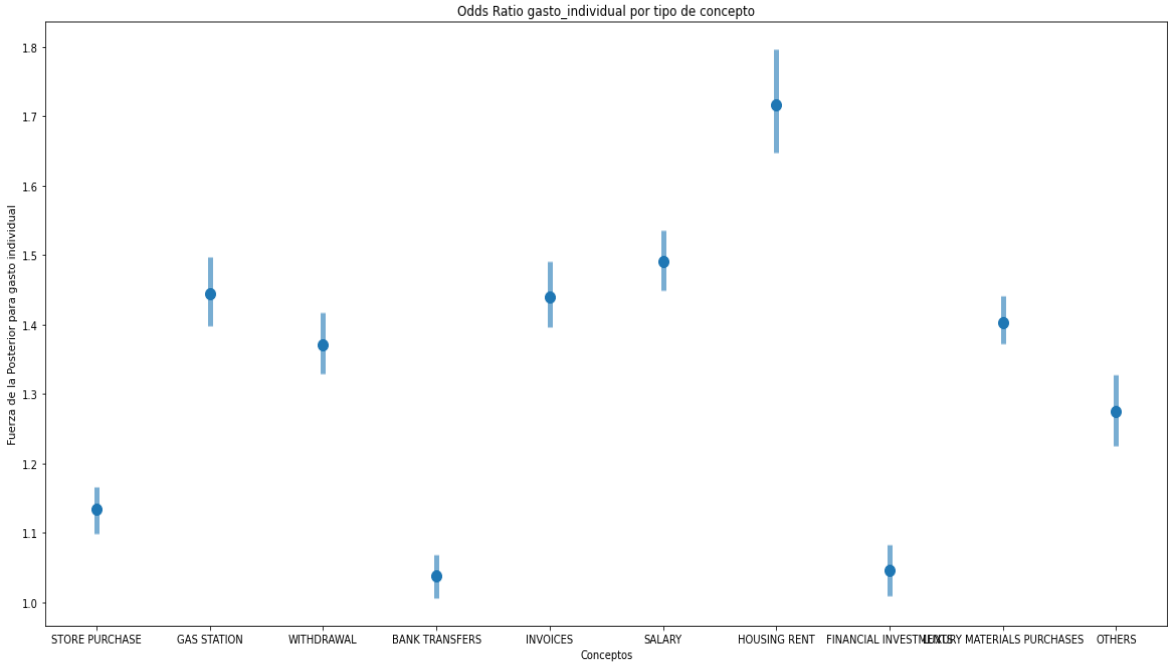
Se agregan dos URLs de la página web oficial del foro de PyMC3 que hablan sobre este error:

https://discourse.pymc.io/t/pymc3-no-longer-importing/6199/16

https://discourse.pymc.io/t/installation-error-vs-tools/4693/6

C Ampliación de Figuras

Gráfica de los *odds ratio* para la variable de gasto especifico, elaborado de manera propia:



Resumen general de todas las variables de nuestro modelo, elaborado de manera propia:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat	odds_ratio	percentage_effect
mu_intecerpt	-1.354	1.079	-3.114	1.028	0.033	0.023	1437.0	824.0	1.00	0.258205	-74.179463
mu_slope	0.206	0.083	0.046	0.355	0.002	0.001	2516.0	1523.0	1.00	1.228753	22.875320
Intercept	-1.411	0.063	-1.529	-1.297	0.002	0.002	729.0	1314.0	1.00	0.243899	-75.610074
dias_agrupados[0]	-0.030	0.011	-0.051	-0.009	0.000	0.000	4770.0	1321.0	1.00	0.970446	-2.955447
dias_agrupados[1]	0.002	0.010	-0.018	0.021	0.000	0.000	3671.0	1402.0	1.00	1.002002	0.200200
dias_agrupados[2]	0.007	0.014	-0.020	0.033	0.000	0.000	2965.0	852.0	1.01	1.007025	0.702456
dias_agrupados[3]	0.027	0.015	-0.000	0.056	0.000	0.000	4260.0	1441.0	1.00	1.027368	2.736780
dias_agrupados[4]	-0.003	0.012	-0.024	0.020	0.000	0.000	4435.0	1762.0	1.00	0.997004	-0.299550
dias_agrupados[5]	-0.552	0.036	-0.611	-0.479	0.001	0.001	2273.0	876.0	1.00	0.575797	-42.420294
dias_agrupados[6]	0.825	0.056	0.724	0.928	0.001	0.001	3133.0	1684.0	1.00	2.281881	128.188077
dias_agrupados[7]	0.023	0.030	-0.033	0.081	0.001	0.001	3304.0	1152.0	1.00	1.023267	2.326654
dias_agrupados[8]	0.006	0.036	-0.065	0.071	0.000	0.001	5661.0	1622.0	1.00	1.006018	0.601804
dias_agrupados[9]	-0.016	0.016	-0.048	0.012	0.000	0.000	2720.0	743.0	1.00	0.984127	-1.587268
veces[0]	0.158	0.010	0.142	0.178	0.000	0.000	2779.0	1673.0	1.00	1.171166	17.116619
veces[1]	0.334	0.014	0.307	0.361	0.000	0.000	2085.0	837.0	1.00	1.396543	39.654314
veces[2]	0.365	0.016	0.337	0.394	0.000	0.000	3787.0	1747.0	1.00	1.440514	44.051401
veces[3]	0.641	0.022	0.601	0.684	0.000	0.000	2634.0	1378.0	1.00	1.898378	89.837831
veces[4]	0.511	0.014	0.484	0.537	0.000	0.000	4276.0	1679.0	1.00	1.666957	66.695732
veces[5]	-0.385	0.084	-0.540	-0.222	0.002	0.001	2862.0	1830.0	1.00	0.680451	-31.954936
veces[6]	0.633	0.062	0.525	0.754	0.001	0.001	4437.0	1903.0	1.00	1.883252	88.325187
veces[7]	1.068	0.049	0.980	1.159	0.001	0.001	2714.0	1807.0	1.00	2.909555	190.955457
veces[8]	0.541	0.055	0.438	0.644	0.001	0.001	2069.0	994.0	1.00	1.717724	71.772373
veces[9]	-0.004	0.019	-0.042	0.030	0.000	0.000	3812.0	1401.0	1.00	0.996008	-0.399201
gasto_ind[0]	0.125	0.016	0.094	0.154	0.001	0.000	923.0	1600.0	1.00	1.133148	13.314845
gasto_ind[1]	0.368	0.018	0.335	0.403	0.001	0.000	763.0	1288.0	1.00	1.444842	44.484204
gasto_ind[2]	0.315	0.017	0.284	0.349	0.001	0.000	884.0	1249.0	1.00	1.370259	37.025931
gasto_ind[3]	0.037	0.016	0.006	0.065	0.001	0.000	1013.0	1495.0	1.00	1.037693	3.769302
gasto_ind[4]	0.364	0.017	0.334	0.399	0.001	0.000	855.0	1407.0	1.00	1.439074	43.907421
gasto_ind[5]	0.399	0.015	0.371	0.429	0.001	0.000	702.0	753.0	1.00	1.490334	49.033362
gasto_ind[6]	0.542	0.023	0.500	0.586	0.001	0.000	1537.0	1555.0	1.00	1.719442	71.944231
gasto_ind[7]	0.045	0.019	0.009	0.079	0.001	0.000	1311.0	1441.0	1.00	1.046028	4.602786
gasto_ind[8]	0.339	0.014	0.317	0.366	0.000	0.000	1166.0	1036.0	1.00	1.403543	40.354335
gasto_ind[9]	0.243	0.022	0.203	0.283	0.001	0.000	1092.0	1255.0	1.00	1.275069	27.506862
cuenta_en_negativo	0.718	0.049	0.625	0.806	0.001	0.001	3863.0	1913.0	1.00	2.050328	105.032845
años_antiguedad_cuenta	-0.209	0.003	-0.215	-0.204	0.000	0.000	4225.0	1797.0	1.00	0.811395	-18.860476
años_persona	-0.491	0.008	-0.505	-0.476	0.000	0.000	5062.0	1499.0	1.00	0.612014	-38.798593
cuentas_persona_en_España	0.466	0.008	0.451	0.480	0.000	0.000	4171.0	1575.0	1.00	1.593607	59.360700
tipo_banco_involucrado[0]	0.786	0.024	0.740	0.832	0.000	0.000	7228.0	1623.0	1.01	2.194600	119.460044
tipo_banco_involucrado[1]	1.289	0.052	1.192	1.381	0.001	0.001	2558.0	1680.0	1.00	3.629156	262.915558
dispone_credito	-1.641	0.045	-1.721	-1.552	0.001	0.000	4950.0	1437.0	1.00	0.193786	-80.621384
nivel_actividad_cuenta[0]	-0.029	0.012	-0.051	-0.006	0.000	0.000	1981.0	845.0	1.00	0.971416	-2.858354
nivel_actividad_cuenta[1]	0.233	0.029	0.178	0.284	0.001	0.000	2269.0	1010.0	1.00	1.262381	26.238148
sigma_intercept	0.849	0.803	0.024	2.312	0.029	0.020	675.0	526.0	1.00	2.337308	133.730837
sigma_slope	0.508	0.058	0.402	0.616	0.001	0.001	4592.0	1493.0	1.00	1.661964	66.196394

D Referencias interesantes sobre la realización del programa

Para el estudio y la realización del programa, se ha ayudado del foro oficial de PyMC3 (https://discourse.pymc.io/) en el cual se han hecho preguntas referentes al proyecto en la que distintos desarrolladores del framework PyMC3 han podido contestar. Ha servido de gran ayuda para la comprensión del funcionamiento de PyMC3. Se adjuntan algunas de ellas a continuación:

- https://discourse.pymc.io/t/how-to-save-my-trace-in-the-system/7222/6
- https://discourse.pymc.io/t/how-to-limit-a-geometric-discrete-distribution-in-a-range/4537/7
- https://discourse.pymc.io/t/general-doubt-about-problems-when-sampling-a-model/7097/9
- https://discourse.pymc.io/t/probability-of-a-certain-value-in-a-discrete-distribution/4570
- https://discourse.pymc.io/t/slow-sampling-in-a-hierarchical-logistic-regression-example/5061/9

E Código relevante del programa

Carga de las librerías y funciones necesarias:

```
import pymc3 as pm
import numpy as np
import pandas as pd
import scipy as sc
import seaborn as sns
import matplotlib.pyplot as plt
import arviz as az
import theano as tt
from arviz.plots.plot_utils import xarray_var_iter
import itertools
from IPython.core.pylabtools import figsize
from sklearn.metrics import accuracy_score, f1_score,
confusion_matrix

from Funciones_Programa import compare_hist
from Funciones_Programa import dia_group
```

Ejemplo de definición de todas las características para el tipo de concepto de compras en tiendas en el proceso de la realización de la BBDD sintética:

```
# STORE PURCHASE

features[0:22000, 0] = np.concatenate([[i] * 22000 for i in
range(1)])
y_data[18000:22000] = np.ones(4000)

mu_norm = 46
sigma_norm = 50
nivel_min_norm = 1
nivel_max_norm = 120
```

```
mu_crim = 46
sigma_crim = 50
nivel_min_crim = 1
nivel_max_crim = 120



features[0:18000,   1]   =   np.random.randint(low=1,   high=31,
size=18000)
features[0:18000, 2] = np.random.poisson(lam=3.5, size=18000)
features[0:18000,  3]  =  sc.stats.truncnorm.rvs((nivel_min_norm  -
mu_norm) / sigma_norm,
                                                 (nivel_max_norm    -
mu_norm) / sigma_norm,
                                                 loc=mu_norm,

scale=sigma_norm,size=18000)

features[18000:22000,   1]   =   np.random.randint(low=1,   high=31,
size=4000)
features[18000:22000, 2] = np.random.poisson(lam=4.0, size=4000)
features[18000:22000, 3] = sc.stats.truncnorm.rvs((nivel_min_crim -
mu_crim) / sigma_crim,
                                                  (nivel_max_crim
- mu_crim) / sigma_crim,
                                                  loc=mu_crim,

scale=sigma_crim,size=4000)
```

Definición del modelo PyMC:

```
with pm.Model(coords=coords) as
cuenta_cliente_hierarchical_centered:

     # constant data
     cat_per_concept = pm.Data("cat_per_concepto", cat_concepto,
     dims="total")
     cat_per_type = pm.Data("cat_per_type", cat, dims="total")

     # Tanto intercept como slope se generan de manera
     jerarquerizada
     mu_intercept = pm.Normal("mu_intecerpt", mu=0., sd=10.)
     sigma_intercept = pm.Exponential('sigma_intercept', lam=1.2)

     mu_slope = pm.Normal("mu_slope", mu=0., sd=10.)
     sigma_slope = pm.Exponential('sigma_slope', lam=1.2)

     intercept = pm.Normal('intercept', mu=mu_intercept,
     sigma=sigma_intercept)
     b1 = pm.Normal('dias_agrupados', mu=mu_slope,
     sigma=sigma_slope,dims='concepto')
     b2 = pm.Normal('veces', mu=mu_slope,
     sigma=sigma_slope,dims='concepto')
```

```
    b3 = pm.Normal('gasto_ind', mu=mu_slope,
    sigma=sigma_slope,dims='concepto')
    b4 = pm.Normal('cuenta_en_negativo', mu=mu_slope,
    sigma=sigma_slope)
    b5 = pm.Normal('años_antiguedad_cuenta', mu=mu_slope,
    sigma=sigma_slope)
    b6 = pm.Normal('años_persona', mu=mu_slope,
    sigma=sigma_slope)
    b7 = pm.Normal('cuentas_persona_en_España', mu=mu_slope,
    sigma=sigma_slope)
    b8 = pm.Normal('tipo_banco_involucrado', mu=mu_slope,
    sigma=sigma_slope, dims='tipo_transaccion')
    b9 = pm.Normal('dispone_credito', mu=mu_slope,
    sigma=sigma_slope)
    b10 = pm.Normal('nivel_actividad_cuenta', mu=mu_slope,
    sigma=sigma_slope, dims='tipo_transaccion')

    #Important line inside the model
    p = pm.math.sigmoid(intercept +
          b1[cat_concepto]*np.array(data_modelo['dia_agrupado'])
          +
          b2[cat_concepto]*np.array(data_modelo['veces_realizada_
          al_mes']) +
          b3[cat_concepto]*np.array(data_modelo['gasto_individual
          _transformed']) +
          b4*np.array(data_modelo['cuenta_en_negativo']) +
          b5*np.array(data_modelo['antiguedad_cuenta']) +
          b6*np.array(data_modelo['edad_transformed']) +
          b7*np.array(data_modelo['numero_cuentas_en_España']) +
          b8[cat]*np.array(data_modelo['tipo_banco_involucrado'])
          +
          b9*np.array(data_modelo['dispone_de_credito']) +
          b10[cat]*np.array(data_modelo['nivel_actividad_cuenta')
          )

    like = pm.Bernoulli('like', p, observed=y_data)
```

Parte del muestreo del modelo (inferencia bayesiana):

```
with cuenta_cliente_hierarchical_centered:
    idata = pm.sample(1200,
                      tune=1200,
                      chains=2,
                      return_inferencedata=True,
                      init="adapt_diag",
                      random_seed=RANDOM_SEED,
                      target_accept=0.96)
```

Muestra de las gráficas de resultados posteriores:

```
var_names=['dias_agrupados', 'veces', 'gasto_ind']
```

```
lines =
list(xarray_var_iter(idata.posterior[var_names].mean(dim=('chain',
'draw'))))
az.plot_trace(idata, divergences=None, var_names=var_names,
lines=lines, combined=True, compact=True);

var_names=['cuenta_en_negativo', 'años_antiguedad_cuenta',
'años_persona',
                        'cuentas_persona_en_España',
'tipo_banco_involucrado', 'dispone_credito',
                        'nivel_actividad_cuenta']

lines =
list(xarray_var_iter(idata.posterior[var_names].mean(dim=('chain',
'draw'))))

az.plot_trace(idata,
              divergences=None,
              var_names=var_names,
              lines=lines,
             combined=True);
```